Project Report

Text Data Processing and Classification Using Attention Mechanism

Submitted by

Niraj Khapne (Roll No. 27)
Shreya Kini (Roll No. 28)
Sashank Mishra (Roll No. 39)
Pradeep Patwa (Roll No. 47)

in partial fulfillment for the award of the degree

BACHELOR OF ENGINEERING

in
Electronics & Telecommunication Engineering

Under the Guidance of

Mr. Santosh Chapaneri

St. Francis Institute of Technology, Mumbai
University of Mumbai
2021-2022

CERTIFICATE

This is to certify that Niraj Khapne, Shreya Kini, Sashank Mishra, and Pradeep Patwa are the bonafide students of St. Francis Institute of Technology, Mumbai. They have successfully carried out the project titled "Text Data Processing and Classification Using Attention Mechanism" in partial fulfillment of the requirement of B. E. Degree in Electronics and Telecommunication Engineering of Mumbai University during the academic year 2021-2022. The work has not been presented elsewhere for the award of any other degree or diploma prior to this.

(Mr. Santosh Chapaneri)
Internal Guide

Internal Examiner

External Examiner

(Dr. Gautam Shah)
EXTC HOD

(Dr. Sincy George)
Principal

Project Report Approval for B.E.

This project entitled ***'Text Data Processing and Classification Using Attention Mechanism'*** by **Niraj Khapne, Shreya Kini, Sashank Mishra, and Pradeep Patwa** is approved for the degree of Bachelor of Engineering in Electronics and Telecommunication from University of Mumbai.

Examiners

1. - - - - - - - - - - - - - - - - - - -

2. - - - - - - - - - - - - - - - - - - -

Date:

Place:

ACKNOWLEDGEMENT

We are thankful to a number of individuals who have contributed towards our final year project and without their help; it would not have been possible. Firstly, we offer our sincere thanks to our project guide, Mr. Santosh Chapaneri for his constant and timely help and guidance throughout our preparation.

We are grateful to all project co-ordinators for their valuable inputs to our project. We are also grateful to the college authorities and the entire faculty for their support in providing us with the facilities required throughout this semester.

We are also highly grateful to Dr. Gautam A. Shah, Head of Department (EXTC), Principal, Dr. Sincy George, and Director Bro. Jose Thuruthiyil for providing the facilities, a conducive environment and encouragement.

Signatures of all the students in the group

(Niraj Khapne)

(Shreya Kini)

(Sashank Mishra)

(Pradeep Patwa)

Declaration

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included; we have adequately cited and referenced the original sources. We also declare that we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in this submission. We understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

Signatures of all the students in the group

(Niraj Khapne)

(Shreya Kini)

(Sashank Mishra)

(Pradeep Patwa)

ABSTRACT

According to reports nearly 80% of the information available today is in unstructured form. The messy nature of text's sorting and organizing process is pretty time consuming and difficult, there comes the text classification with ML reducing human efforts and with certain predefined algorithms and scripts all the job is done let that be in analyzing survey reports, sales reports,etc. Typical RNN/LSTM neural networks consider all words in the text data as important; however, this may not always be true. Hence, we will use the Attention mechanism so that the algorithm focuses on specific words resulting in improved performance. Here we are using sequence transduction models based on complex recurrent or convolutional neural networks that include an encoder and a decoder. We are using Transformer network architecture which is solely based on an attention mechanism. The transformer allows significantly more parallelization in sequential data and requires significantly less time to train.

Contents

Approval ii

Acknowledgement iii

Declaration iv

List of Figures viii

List of Tables ix

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 1
- 1.3 Scope of Project . . . 2
- 1.4 Organization of Project . . . 2

2 Literature Survey 3
- 2.1 Feed-Forward Neural Networks [1]: . . . 3
- 2.2 RNN-Based Models [2]: . . . 5
- 2.3 CNN-Based Models [3]: . . . 7
- 2.4 Models with Attention Mechanism [4]: . . . 10
- 2.5 Graph Neural Networks [5]: . . . 12
- 2.6 Siamese Neural Networks [6]: . . . 14

3 Preliminaries 16
- 3.1 Artificial Neural Network . . . 16
- 3.2 Fully Connected Network . . . 18
- 3.3 Training . . . 19
- 3.4 Backpropagation . . . 20
- 3.5 Loss Functions . . . 21
- 3.6 Gradient Descent . . . 23
- 3.7 Activation Functions . . . 24
- 3.8 Optimization . . . 26

4 Proposed Methodology 28
- 4.1 Text Preprocessing : . . . 29
- 4.2 Feature engineering . . . 31
- 4.3 Attention mechanism: . . . 34
- 4.4 Transformer Architecture: . . . 42

4.4.1 Embedding & Positional Encoding: 43
4.4.2 Model Computations 44
4.4.3 SoftMax: 47
4.5 Dataset: 48
4.5.1 Exploratory Data Analysis (EDA) : 49
4.5.2 Results and Discussion : 49

5 Conclusion **51**

Bibliography **52**

List of Figures

2.1 The architecture of the Deep Average Network (DAN). 3
2.2 The doc2vec model 4
2.3 Traditional Structure of RNN 5
2.4 A chain-structured LSTM network and a tree-structured LSTM network with arbitrary branching factor. Here *xi* and *yi* denote the input and output of each cell. 6
2.5 The architecture of DCNN model 7
2.6 The architecture of a sample CNN model for text classification. 8
2.7 The architecture of a character-level CNN model 9
2.8 (a) The architecture of attentive pooling networks. (b) The architecture of label-text matching model. 10
2.9 The architecture of GNN used by Peng et al. 12
2.10 The architecture of GCNN. 13
2.11 The architecture of a DSSM. 14
2.12 The architecture of the Siamese model 15
2.13 The architecture of the Siamese model 15

3.1 Artificial Neural Network. 16
3.2 Fully connected neural network. 18
3.3 Neural network as a black box. 19
3.4 Backpropagation. 20
3.5 Hard decision from softmax probailities. 21
3.6 Schematic of Gradient Descent. 23
3.7 Sigmoid function. 24
3.8 ReLU function. 25

4.1 Flowchart 28
4.2 Tokenization 29
4.3 Self Attention 35
4.4 Multi-head Attention 37
4.5 Reshaping 39
4.6 Computing Attention Scores 40
4.7 Transformer Model Architecture 42
4.8 Model Computations 44
4.9 Model Computations 45
4.10 model summary 50

List of Tables

4.1 classification report 49

List of Abbreviations

DL	Deep Learning
NLP	Natural Language Processing
MLP	Multi-Layer Perceptrons
DAN	Deep Average Network
RNN)	Recurrent Neural Network
LSTM	Long short-term memory
SVM	Support Vector Machine
CE	Cross Entropy
ADAM	Adaptive moment estimation
MT-LSTM	Multi-Timescale Long short-term memory
Bi-LSTM	Bidirectional Long short-term memory
DNN	Deep Neural Network
CNN	Convolutional Neural Network
VDCNN	Very Deep Convolutional Neural Network
GNNs)	Graph Neural Networks
ANN	Artificial Neural Network
IMDb	Internet Movie Database
EDA	Exploratory Data Analysis
ReLU	Rectified Linear Unit
SGC	Simple Graph Convolution
TFIDF	Term frequency Inverse Document Frequency
Q	Query
K	Key
V	Value

Chapter 1

Introduction

The way text is represented has a big impact on how well text classification systems work. Text classification is the process of determining text categories based on the text content within a given classification system. Text classification is a fundamental task in natural language processing that has a wide range of applications, including sentiment analysis, topic labelling, spam detection, and intent identification.Natural language processing (NLP) involves computational processing and understanding of the natural/human languages. It involves various tasks that rely on various statistics and data-driven computation techniques. One of the important tasks in NLP is text classification. Automatic text classification uses machine learning, natural language processing (NLP), and other AI-guided approaches to categorise text more quickly, efficiently, and accurately. NLP has had a lot of success in the industries of healthcare, media, finance, and human resources. We are using attention mechanism which is a component of a neural architecture that allows for dynamically highlighting relevant parts of the input data, which is often a sequence of textual elements in NLP.

1.1 Motivation

It's estimated that around 80% of all information is unstructured, with text being one of the most common types of unstructured data. Because of the messy nature of text, analyzing, understanding, organizing, and sorting through text data is hard and time-consuming, so most companies fail to use it to its full potential. This is where text classification with machine learning comes in. Using text classifiers, companies can automatically structure all manner of relevant text, from emails, legal documents, social media, chatbots, surveys, and more in a fast and cost-effective way. This allows companies to save time analyzing text data, automate business processes, and make data-driven business decisions.Manually evaluating and arranging data is time-consuming and inaccurate. Machine learning can evaluate millions of surveys, comments, emails, and other documents at a fraction

1.2 Problem Statement

With the exponential growth of text data in the industry and on the Internet, it's becoming increasingly important for us to extract meaningful data from such unstructured data, so we utilise text classification. Here, we are doing sentiment analysis of movie review,

by taking IMDB dataset as input then we are doing text preprocessing and feature engineering of the input data then we pass it to our model to be trained then after training with 80%, we test it with 20% of dataset.

1.3 Scope of Project

The method of evaluating and extracting information from textual data is already being used in fields including marketing, product management, academics, and governance. For a long time, we've relied on text classification to make things easier for us. Text categorization is exemplified by the classification of books in libraries and the segmentation of news items. By incorporating AI technology, the process becomes more automated and straightforward, requiring less manual labour. The idea of employing artificial intelligence to classify text has been around for a while.

1.4 Organization of Project

- Literature Survey: In the initial stages of the conceptualization of this project, we present a survey of literature to study the work that has already been done in the field of Natural Language Processing(NLP).
- Study of Machine Learning: A brief introduction to various types of machine learning and their parameters is described in this section.
- Study of Supervised Learning: Supervised learning is the machine learning task of learning a function that maps an input to an output based on the eg. of input output pairs. Supervised learning algorithms are trained using labelled data sets.
- Study of Recurrent Neural Network : (RNN) unable to learn long range dependencies as they can process more words but it has trouble retaining previous words/steps which create vanishing gradient problems. i.e short term memory.
- Study of Long short-term memory : Due to the limitations of RNNs like not remembering long term dependencies, in practice, we almost always use LSTM to model long term dependencies.
- Study of Attention mechanism : The attention mechanism is a part of a neural architecture that enables dynamically highlighting relevant features of the input data, which, in NLP, is typically a sequence of textual elements. It can be applied directly to the raw input or to its higher level representation.
- Study of Transformer Model : It excels in handling text data which is inherently sequential. Consists of Encoder and Decoder Stacks each consisting of their corresponding embedding layers for respective inputs. It uses self attention by relating every word in Input sequence to every other word.

Chapter 2

Literature Survey

2.1 Feed-Forward Neural Networks [1]:

Feed-forward networks are among the simplest DL models for text representation. Yet, they have achieved high accuracy on many TC benchmarks. These models view text as a bag of words. For each word, they learn a vector representation using an embedding model such as word2vec or Glove, take the vector sum or average of the embeddings as the representation of the text, pass it through one or more feed-forward layers, known as Multi-Layer Perceptrons (MLPs), and then perform classification on the final layer's representation using a classifier such as logistic regression, Naïve Bayes, or SVM . An example of these models is the Deep Average Network (DAN) , whose architecture is shown in Fig. 2.1 Despite its simplicity, DAN outperforms other more sophisticated models which are designed to explicitly learn the compositionality of texts. For example, DAN outperforms syntactic models on datasets with high syntactic variance. Joulin et al.[13] propose a simple and efficient text classifier called fastText. Like DAN, fastText views text as a bag of words. Unlike DAN, fastText uses a bag of n-grams as additional features to capture local word order information. This turns out to be very efficient in practice, achieving comparable results to the methods that explicitly use the word order.

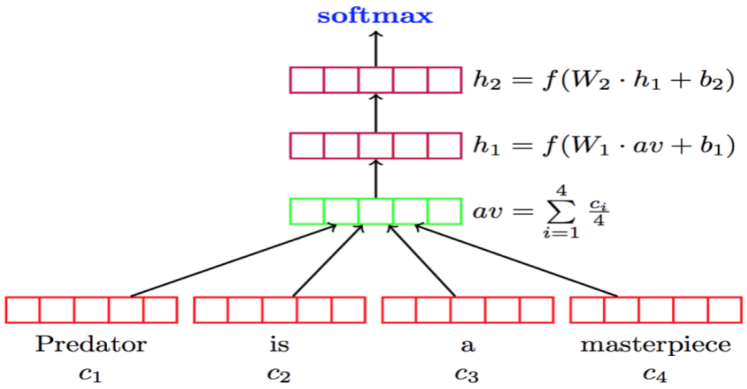

Figure 2.1: The architecture of the Deep Average Network (DAN).

Le and Mikolov [14]propose doc2vec, which uses an unsupervised algorithm to learn fixed-length feature representations of variable-length pieces of texts, such as sentences, paragraphs, and documents. As shown in Fig, the architecture of doc2vec is similar to that

of the Continuous Bag of Words (CBOW) model. The only difference is the additional paragraph token that is mapped to a paragraph vector via matrix.

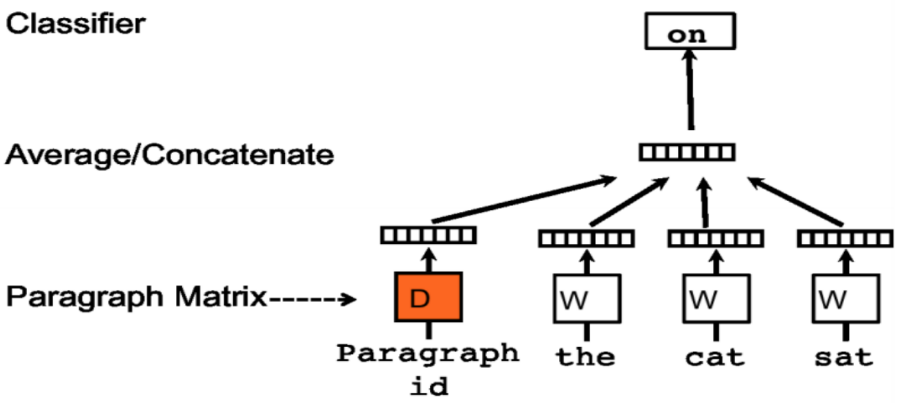

Figure 2.2: The doc2vec model

In doc2vec, the concatenation or average of this vector with a context of three words is used to predict the fourth word. The paragraph vector represents the missing information from the current context and can act as a memory of the topic of the paragraph. After being trained, the paragraph vector is used as features for the paragraph (e.g., in lieu of or in addition to BoW), and fed to a classifier for prediction. Doc2vec achieves new state of the art results on several TC tasks when it is published.

2.2 RNN-Based Models [2]:

RNN-based models view text as a sequence of words, and are intended to capture word dependencies and text structures for TC. However, vanilla RNN models do not perform well, and often underperform feed-forward neural networks. Among many variants to RNNs, Long Short-Term Memory (LSTM) is the most popular architecture, which is designed to better capture long term dependencies. LSTM addresses the gradient vanishing or exploding problems suffered by the vanilla RNNs by introducing a memory cell to remember values over arbitrary time intervals, and three gates (input gate, output gate, forget gate) to regulate the flow of information into and out of the cell. There have been works on improving RNNs and LSTM models for TC by capturing richer information, such as tree structures of natural language, long-span word relations in text, document topics, and so on.

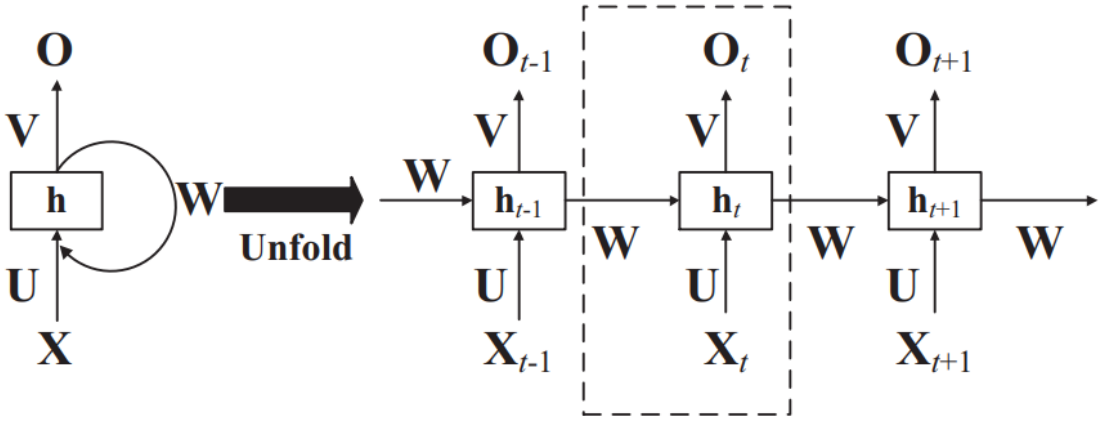

Figure 2.3: Traditional Structure of RNN

$$h_t = f(Wh_{t-1} + Ux_t + b_h) and o_t = softmax(Vh_t + b_0) \tag{2.1}$$

Tai et al [8]. develop a Tree-LSTM model, a generalization of LSTM to tree-structured network typologies, to learn rich semantic representations. The authors argue that Tree-LSTM is a better model than the chain-structured LSTM for NLP tasks because natural language exhibits syntactic properties that would naturally combine words to phrases. They validate the effectiveness of Tree-LSTM on two tasks: sentiment classification and predicting the semantic relatedness of two sentences. The architectures of these models are shown in Fig.2.4 . Zhu et al.[15] also extend the chain-structured LSTM to tree structures, using a memory cell to store the history of multiple child

To model long-span word relations for machine reading, Cheng et al. [16] augment the LSTM architecture with a memory network in place of a single memory cell. This enables adaptive memory usage during recurrence with neural attention, offering a way to weakly induce relations among tokens. This model achieves promising results on language modeling, sentiment analysis, and NLI. The Multi-Timescale LSTM (MT-LSTM) neural network is also designed to model long texts, such as sentences and documents, by capturing valuable information with different timescales. MT-LSTM partitions the hidden states of a standard LSTM model into several groups. Each group is activated and updated at different time periods. Thus, MT-LSTM can model very long documents.

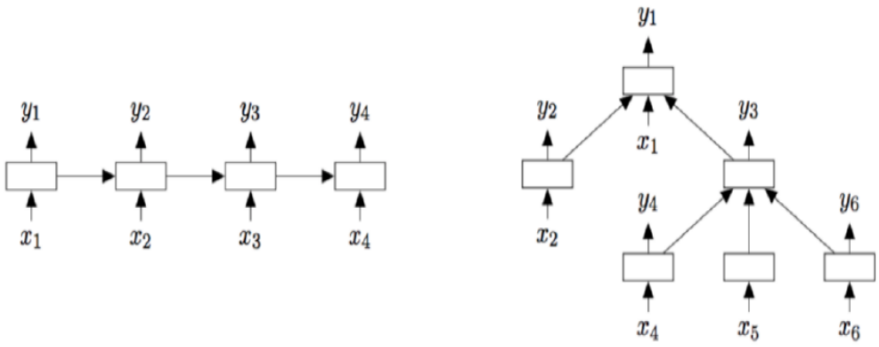

Figure 2.4: A chain-structured LSTM network and a tree-structured LSTM network with arbitrary branching factor. Here *xi* and *yi* denote the input and output of each cell.

MT-LSTM is reported to outperform a set of baselines, including the models based on LSTM and RNN, on TC

RNNs are good at capturing the local structure of a word sequence, but face difficulties remembering long-range dependencies. In contrast, latent topic models are able to capture the global semantic structure of a document but do not account for word ordering. Dieng et al [2]. propose a TopicRNN model to integrate the merits of RNNs and latent topic models. It captures local (syntactic) dependencies using RNNs and global (semantic) dependencies using latent topics. TopicRNN is reported to outperform RNN baselines for sentiment analysis. There are other interesting RNN-based models. Liu et al [16]. use multi-task learning to train RNNs to leverage labeled training data from multiple related tasks. Johnson and Rie explore a text region embedding method using LSTM. Zhou et al [17]. integrate a Bidirectional-LSTM (Bi-LSTM) model with two-dimensional maxpooling to capture text features. Wang et al. propose a bilateral multi-perspective matching model under the "matching-aggregation" framework. he also explore semantic matching using multiple positional sentence representations generated by a bi-directional LSMT model. It is worth noting that RNNs belong to a broad category of DNNs, known as recursive neural networks. A recursive neural network applies the same set of weights recursively over a structure input to produce a structured prediction or a vector representation over variable-size input. Whereas RNNs are recursive neural networks with a linear chain structure input, there are recursive neural networks that operate on hierarchical structures, such as parse trees of natural language sentences, combining child representations into parent representations. RNNs are the most popular recursive neural networks for TC because they are effective and easy to use – they view text as a sequence of tokens without requiring additional structure labels such as parse trees.

2.3 CNN-Based Models [3]:

RNN's are trained to recognize patterns across time, whereas CNNs learn to recognize patterns across space.RNNs work well for the NLP tasks such as POS tagging or QA where the comprehension of long-range semantics is required, while CNNs work well where detecting local and position-invariant patterns is important. These patterns could be key phrases that express a particular sentiment like "I like" or a topic like "endangered species".

Thus, CNN's have become one of the most popular model architectures for TC. One of the first CNN-based models for TC is proposed by Kalchbrenner et al. The model uses dynamic k-max-pooling, and is called the Dynamic CNN (DCNN). As illustrated in Fig, the first layer of DCNN constructs a sentence matrix using the embedding for each word in the sentence. Then a convolutional architecture that alternates wide convolutional layers with dynamic pooling layers given by dynamic kmax-pooling is used to generate a feature map over the sentence that is capable of explicitly capturing short and long-range relations of words and phrases. The pooling parameter k can be dynamically chosen depending on the sentence size and the level in the convolution hierarchy.

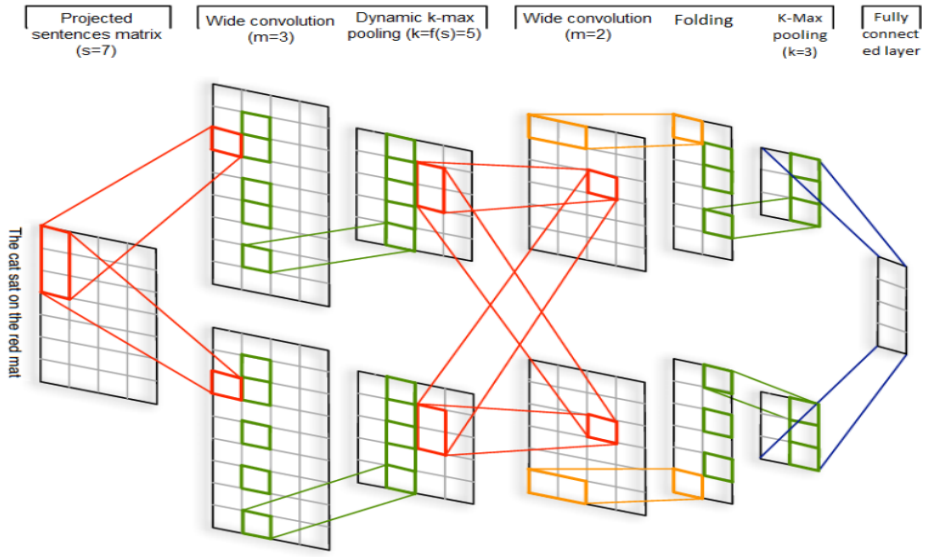

Figure 2.5: The architecture of DCNN model

Later, Kim proposes a much simpler CNN-based model than DCNN for TC. As shown in Fig , Kim's model uses only one layer of convolution on top of the word vectors obtained from an unsupervised neural language model i.e., word2vec. Kim also compares four different approaches to learning word embeddings:

(1) CNN-rand, where all word embeddings are randomly initialized and then modified during training;

(2) CNN-static, where the pre-trained word2vec embeddings are used and stay fixed during model training;

(3) CNN-non-static, where the word2vec embeddings are fine-tuned during training for each task; and

(4) CNN-multi-channel, where two sets of word embedding vectors are used, both are initialized using word2vec, with one updated during model training while the other fixed.

These CNN-based models are reported to improve upon the state of the art on sentiment analysis and question classification.

There have been efforts of improving the architectures of CNN-based models of. Liu et al. propose a new CNN-based model that makes two modifications to the architecture of Kim-CNN. First, a dynamic max-pooling scheme is adopted to capture more fine-grained features from different regions of the document. Second, a hidden bottleneck layer is inserted between the pooling and output layers to learn compact document representations to reduce model size and boost model performance. In, instead of using pre-trained low-dimensional word vectors as input to CNNs, the authors directly apply CNNs to high-dimensional text data to learn the embeddings of small text regions for classification. Character-level CNNs have also been explored for TC. One of the first such models is proposed by Zhang et al. As illustrated in Fig. 2.6 , the model takes as input the characters in a fixed-sized, encoded as one-hot vectors, passes them through a deep CNN model that consists of six convolutional layers with pooling operations and three fully connected layers. Prusa et al. present an approach to encoding text using CNNs that greatly reduces memory consumption and training time required to learn character-level text representations. This approach scales well with alphabet size, allowing to preserve more information from the original text to enhance classification performance.

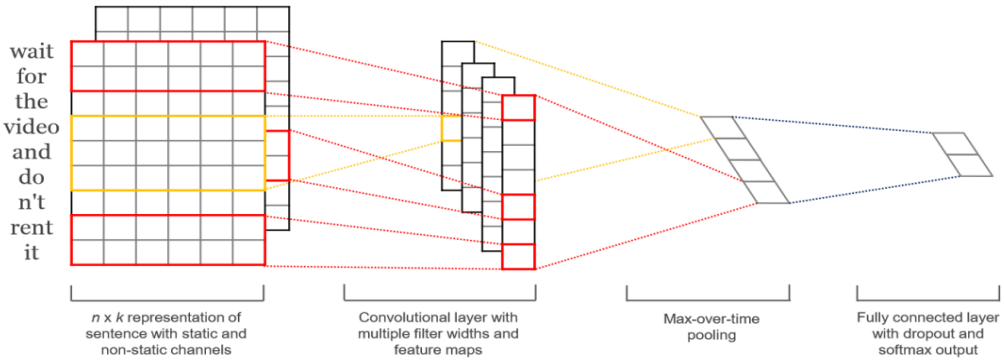

Figure 2.6: The architecture of a sample CNN model for text classification.

There are studies on investigating the impact of word embeddings and CNN architectures on model performance. Inspired by VGG and ResNets, Conneau et al. present a Very Deep CNN (VDCNN) model for text processing. It operates directly at the character level and uses only small convolutions and pooling operations. This study shows that the performance of VDCNN improves with the increase of the depth. Duque et al. modify the structure of VDCNN to fit mobile platforms' constraints without much performance degradation. They are able to compress the model size by 10x to 20x with an accuracy loss, show that deep models indeed outperform shallow models when the text input is represented as a sequence of characters.

However, a simple shallow-and-wide network outperforms deep models such as DenseNet with word inputs. Guo et al. study the impact of word embedding and propose to use weighted word embeddings via a multi-channel CNN model. Zhang et al. examine the impact of different word embedding methods and pooling mechanisms, and find that using non-static word2vec and GloVe outperforms one-hot vectors, and that max-pooling consistently outperforms other pooling methods. There are other interesting CNN-based

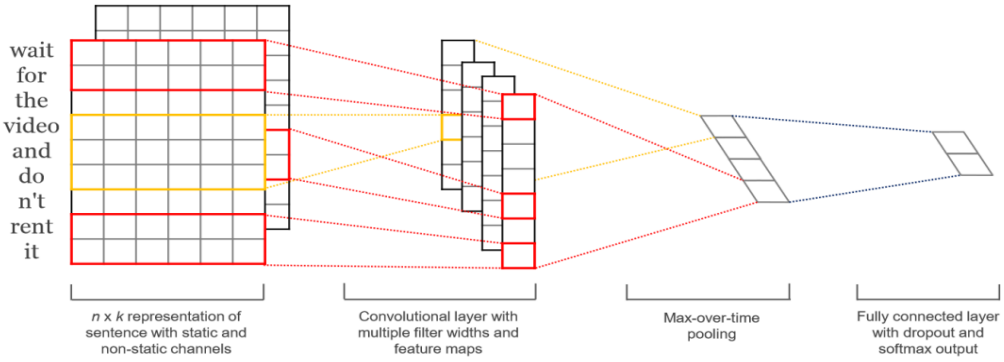

Figure 2.7: The architecture of a character-level CNN model

models. Mou et al. present a tree-based CNN to capture sentencelevel semantics. Pang et al. cast text matching as the image recognition task, and use multi-layer CNNs to identify salient n-gram patterns. Wang et al. propose a CNN-based model that combines explicit and implicit representations of short text for TC. There is also a growing interest in applying CNNs to biomedical text classification .

2.4 Models with Attention Mechanism [4]:

Attention is motivated by how we pay visual attention to different regions of an image or correlate words in one sentence. Attention becomes an increasingly popular concept and useful tool in developing DL models for NLP.

In a nutshell, attention in language models can be interpreted as a vector of importance weights. In order to predict a word in a sentence, we estimate using the attention vector how strongly it is correlated with, or "attends to", other words and take the sum of their values weighted by the attention vector as the approximation of the target. This section reviews some of the most prominent attention models which create new state of the arts on TC tasks, when they are published. Yang et al. propose a hierarchical attention network for text classification. This model has two distinctive characteristics:

(1) a hierarchical structure that mirrors the hierarchical structure of documents, and

(2) two levels of attention mechanisms applied at the word and sentence-level, enabling it to attend differentially to more and less important content when constructing the document representation. This model outperforms previous methods by a substantial margin on six TC tasks. Zhou et al. extend the hierarchical attention model to cross-lingual sentiment classification. In each language, a LSTM network is used to model the documents.

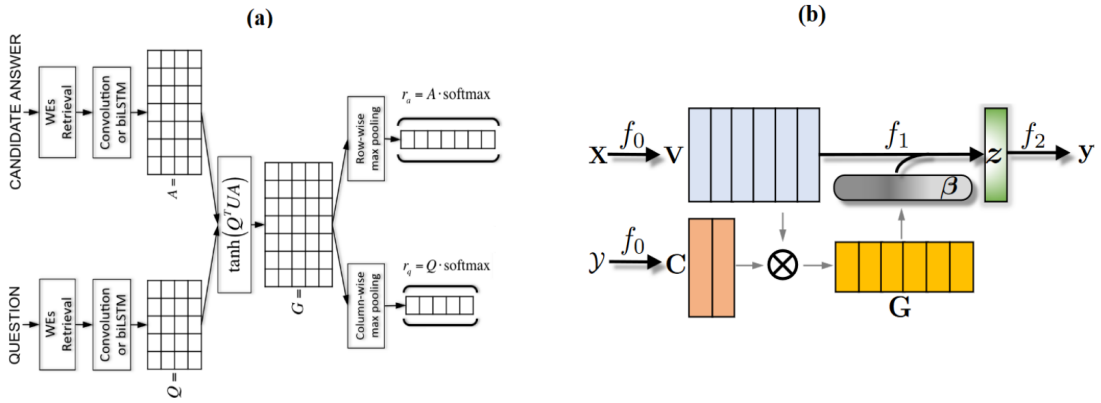

Figure 2.8: (a) The architecture of attentive pooling networks. (b) The architecture of label-text matching model.

Then, classification is achieved by using a hierarchical attention mechanism, where the sentence-level attention model learns which sentences of a document are more important for determining the overall sentiment. while the word-level attention model learns which words in each sentence are decisive. Shen et al. present a directional self-attention network for RNN/CNN-free language understanding, where the attention between elements from input sequence(s) is directional and multi-dimensional.

A light-weight neural net is used to learn sentence embedding, solely based on the proposed attention without any RNN/CNN structure. Liu et al. present a LSTM model with inner-attention for NLI. This model uses a two-stage process to encode a sentence. Firstly, average pooling is used over word-level Bi-LSTMs to generate a first stage sentence representation. Secondly, attention mechanism is employed to replace average pooling on the same sentence for better representations.

The sentence's first-stage representation is used to attend words appeared in itself. Attention models are widely applied to pair-wise ranking or text matching tasks too. Santos et al. propose a two-way attention mechanism, known as Attentive Pooling (AP), for pair-wise ranking. AP enables the pooling layer to be aware of the current input pair (e.g., a question-answer pair), in a way that information from the two input items can directly influence the computation of each other's representations. In addition to learning the representations of the input pair, AP jointly learns a similarity measure over projected segments of the pair, and subsequently derives the corresponding attention vector for each input to guide the pooling.

AP is a general framework independent of the underlying representation learning, and can be applied to both CNNs and RNNs, as illustrated in Fig. Wang et al.view TC as a label-word matching problem: each label is embedded in the same space with the word vector. The authors introduce an attention framework that measures the compatibility of embeddings between text sequences and labels via cosine similarity, as shown in Fig Kim et al.propose a semantic sentence matching approach using a densely-connected recurrent and coattentive network. Similar to DenseNet , each layer of this model uses concatenated information of attentive features as well as hidden features of all the preceding recurrent layers. It enables preserving the original and the co-attentive feature information from the bottom-most word embedding layer to the uppermost recurrent layer. Yin et al. present another attention-based CNN model for sentence pair matching. They examine three attention schemes for integrating mutual influence between sentences into CNNs, so that the representation of each sentence takes into consideration its paired sentence.

These interdependent sentence pair representations are shown to be more powerful than isolated sentence representations, as validated on multiple classification tasks including answer selection, paraphrase identification, and textual entailment. Tan et al. employ multiple attention functions to match sentence pairs under the matching-aggregation framework. Yang et al. introduce an attention-based neural matching model for ranking short answer texts. They adopt value-shared weighting scheme instead of position-shared weighting scheme for combining different matching signals and incorporated question term importance learning using question attention network.

This model achieves promising results on the TREC QA dataset.There are other interesting attention models. Lin et al. used self-attention to extract interpretable sentence embeddings. Wang et al.proposed a densely connected CNN with multi-scale feature attention to produce variable n-gram features. Yamada and Shindo used neural attentive bag-of-entities models to perform TC using entities in a knowledge base. Parikh et al.used attention to decompose a problem into sub-problems that can be solved separately. Chen et al. explored generalized pooling methods to enhance sentence embedding, and proposed a vector-based multi-head attention model. Basiri et al. proposed an attention-based bidirectional CNN-RNN deep model for sentiment analysis

2.5 Graph Neural Networks [5]:

Although natural language texts exhibit a sequential order, they also contain internal graph structures, such as syntactic and semantic parse trees, which define the syntactic and semantic relations among words in sentences. One of the earliest graph-based models developed for NLP is TextRank . The authors propose to represent a natural language text as a graph G(V,E), where V denotes a set of nodes and E set of edges among the nodes.

Depending on the applications at hand, nodes can represent text units of various types, e.g., words, collocations, entire sentences, etc. Similarly, edges can be used to represent different types of relations between any nodes, e.g., lexical or semantic relations, contextual overlap, etc. Modern Graph Neural Networks (GNNs) are developed by extending DL approaches for graph data, such as the text graphs used by TextRank. Deep neural networks, such as CNNs, RNNs and autoencoders, have been generalized over the last few years to handle the complexity of graph data. For example, a 2D convolution of CNNs for image processing is generalized to perform graph convolutions by taking the weighted average of a node's neighborhood information.

Among various types of GNNs, convolutional GNNs, such as Graph Convolutional Networks (GCNs) and their variants, are the most popular ones because they are effective and convenient to compose with other neural networks, and have achieved state of the art results in many applications. GCNs are an efficient variant of CNNs on graphs. GCNs stack layers of learned first-order spectral filters followed by a nonlinear activation function to learn graph representations. A typical application of GNNs in NLP is TC. GNNs utilize the inter-relations of documents or words to infer document labels.

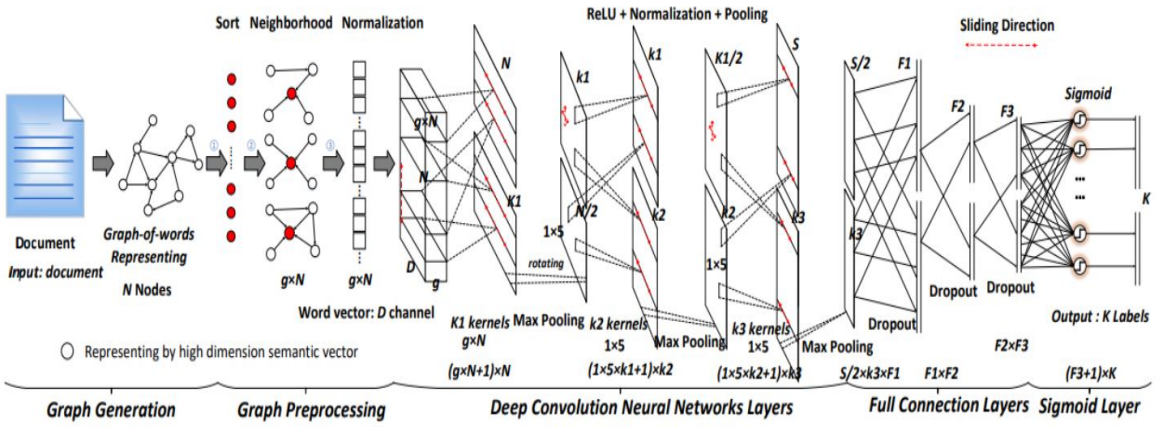

Figure 2.9: The architecture of GNN used by Peng et al.

In what follows, we review some variants of GCNs that are developed for TC. Peng et al. propose a graph-CNN based DL model to first convert text to graph-of-words, and then use graph convolution operations to convolve the word graph, as shown in Fig They show through experiments that the graph-of-words representation of texts has the advantage of capturing non-consecutive and long-distance semantics, and CNN models have the advantage of learning different level of semantics. In , Peng et al. propose a TC model based on hierarchical taxonomy-aware and attentional graph capsule CNNs.

One unique feature of the model is the use of the hierarchical relations among the class labels, which in previous methods are considered independent. Specifically, to leverage

such relations, the authors develop a hierarchical taxonomy embedding method to learn their representations, and define a novel weighted margin loss by incorporating the label representation similarity. Yao et al.use a similar Graph CNN (GCNN) model for TC. They build a single text graph for a corpus based on word co-occurrence and document word relations, then learn a Text Graph Convolutional Network (Text GCN) for the corpus, as shown in Fig.2.10. The Text GCN is initialized with one-hot representation for word and document, and then jointly learns the embeddings for both words and documents, as supervised by the known class labels for documents.

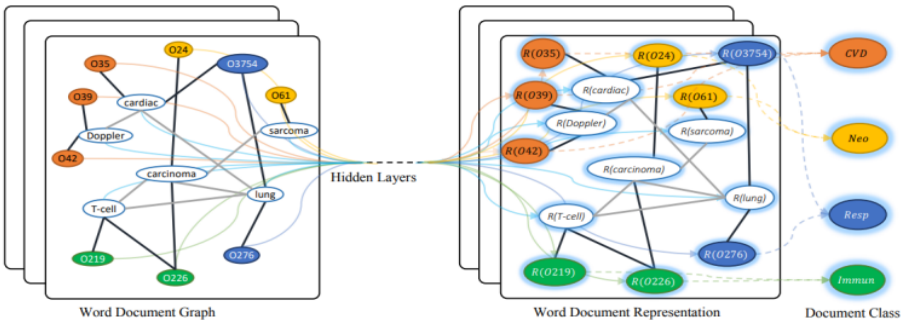

Figure 2.10: The architecture of GCNN.

Building GNNs for a large-scale text corpus is costly. There have been works on reducing the modeling cost by either reducing the model complexity or changing the model training strategy. An example of the former is the Simple Graph Convolution (SGC) model proposed , where a deep convolutional GNN is simplified by repeatedly removing the non-linearities between consecutive layers and collapsing the resulting functions (weight matrices) into a single linear transformation. An example of the latter is the text-level GNN. Instead of building a graph for an entire text corpus, a text-level GNN produces one graph for each text chunk defined by a sliding window on the text corpus so as to reduce the memory consumption during training. Some of the other promising GNN based works include, GraphSage , and contextualized non-local neural network.

2.6 Siamese Neural Networks [6]:

Siamese neural networks (S2Nets) and their DNN variants, known as Deep Structured Semantic Models (DSSMs) , are designed for text matching. The task is fundamental to many NLP applications, such as query-document ranking and answer selection in extractive QA. These tasks can be viewed as special cases of TC. For example, in question-document ranking, we want to classify a document as relevant or irrelevant to a given query.

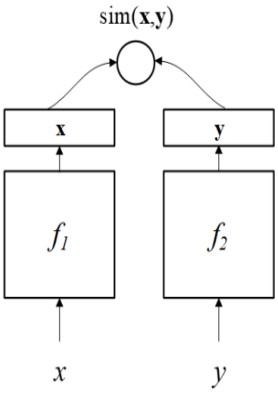

Figure 2.11: The architecture of a DSSM.

As illustrated in Fig , a DSSM (or a S2Net) consists of a pair of DNNs, *f1* and *f2*, which map inputs x and y into corresponding vectors in a common low-dimensional semantic space. Then the similarity of x and y is measured by the cosine distance of the two vectors. While S2Nets assume that *f1* and *f2* share the same architecture and even the same parameters, in DSSMs, *f1* and *f2* can be of different architectures depending on x and y For example, to compute the similarity of an image-text pair, *f1* can be a deep CNN and *f2* an RNN or MLP.

These models can be applied to a wide range of NLP tasks depending on the definition of (x,y). For example, *(x,y)* could be a query-document pair for query-document ranking or a question-answer pair in QA The model parameters theta often optimized using a pair-wise rank loss. Take document ranking as an example. Consider a query x and two candidate documents y

Since texts exhibit a sequential order, it is natural to implement *f1* and *f2* using RNNs or LSTMs to measure the semantic similarity between texts. Fig shows the architecture of the siamese model proposed in , where the two networks use the same LSTM model. Neculoiu et al. present a similar model that uses character-level Bi-LSTMs for *f1* and *f2,* and the cosine function to calculate the similarity. Liu et al. model the interaction of a sentence pair with two coupled-LSTMs.

In addition to RNNs, BOW models and CNNs are also used in S2Nets to represent sentences. For example, He et al. propose a S2Net that uses CNNs to model multi-perspective sentence similarity. Renter et al.propose a Siamese CBOW model which forms a sentence vector representation by averaging the word embeddings of the sentence, and calculates the sentence similarity as cosine similarity between sentence vectors. As BERT becomes the new state of the art sentence embedding model, there have been attempts to building BERT-based S2Nets, such as SBERT and TwinBERT. S2Nets and DSSMs have

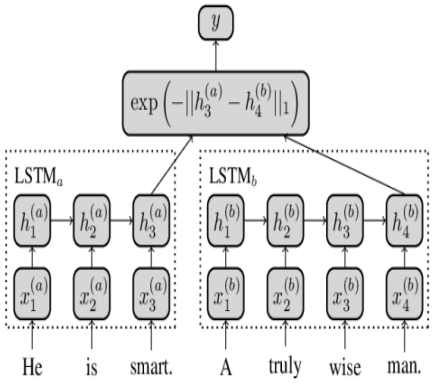

Figure 2.12: The architecture of the Siamese model

been widely used for QA. Das et al. propose a Siamese CNN for QA (SCQA) to measure the semantic similarity between a question and its (candidate) answers.

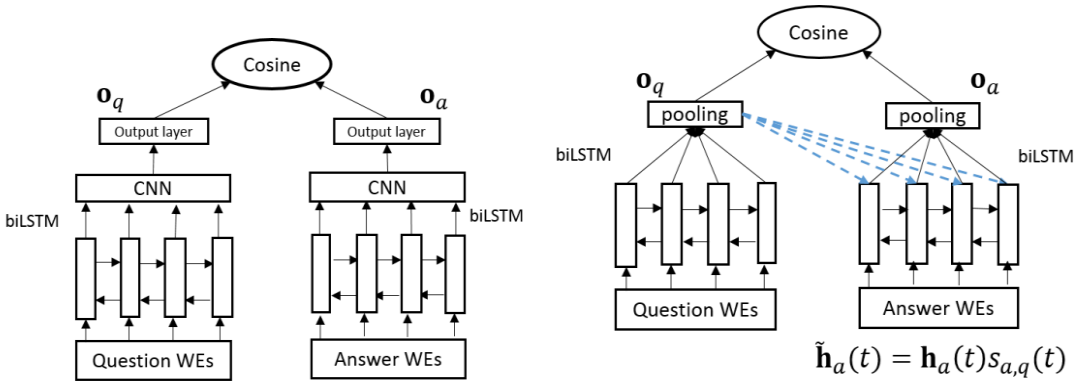

Figure 2.13: The architecture of the Siamese model

To reduce the computational complexity, SCQA uses character-level representations of question-answer pairs. The parameters of SCQA is trained to maximize the semantic similarities between a question and its relevant answers, as Equation 1, where x is a question and y its candidate answer. Tan et al. present a series of siamese neural networks for answer selection. As shown in Fig. 2.13 , these are hybrid models that process text using convolutional, recurrent, and attention neural networks. Other siamese neural networks developed for QA include LSTM-based models for non-factoid answer selection, Hyperbolic representation learning, and QA using a deep similarity neural network.

Chapter 3

Preliminaries

3.1 Artificial Neural Network

Artificial neural networks (ANN) are systems that are used to replicate human brains. A neural network consists of input, hidden and output layers, where the hidden layer transforms the inputs to obtain a particular output that we want. It can be considered as one of the tools which can be helpful to human to extract complex patterns and make the machine to make decisions on their own. The decision making capability can be improved by the self-learning process. Neural networks are also called Multilayer perceptrons which is one of the important parts of artificial intelligence. It is due to backpropagation which allows the networks to adjust their hidden layers in situations where output is not matching as per the expected results. For example, if a network is designed to identify a boy but it has recognized as a girl. Here multilayer network extracts different features until it can recognize as per the expected results.

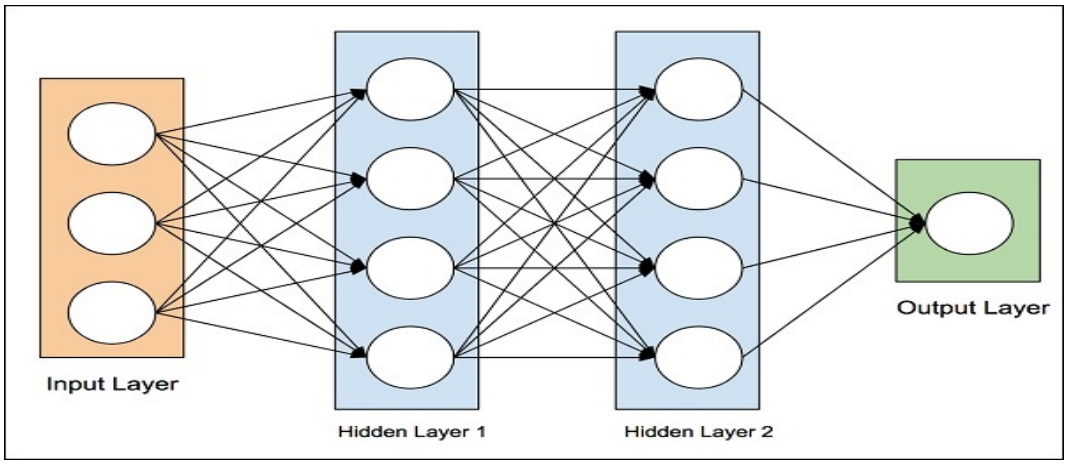

Figure 3.1: Artificial Neural Network.

Input Layer:

The input layer can be in the form of text, images, numbers, etc. Every input neuron represents a variable that has some influence over the output of the neural network.

Hidden Layer:

The hidden layer process the inputs which are obtained by the previous layer. It extracts the required feature from the input which has activation function applied to it. There can be multiple hidden layers in a Neural Network for complex feature extraction.

Output Layer:

The output layer of the neural network collects the output from the hidden layer and if the outcomes are not matched by the desired outputs then by using backpropagation and changing the hyperparameters the required outputs can be obtained.

3.2 Fully Connected Network

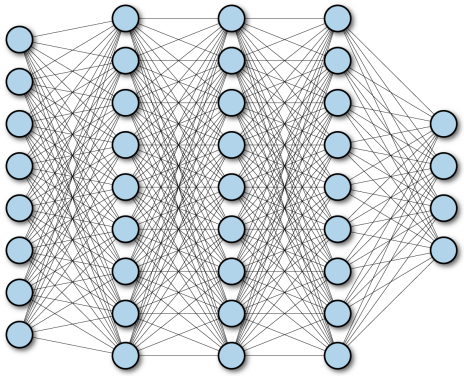

Figure 3.2: Fully connected neural network.

A fully connected neural network consists of a series of fully connected layers. A fully connected layer is a function from R_m to R_n . Each output dimension depends on each input dimension. Let $x \in R_m$ represent the input to a fully connected layer. Let $y_i \; in \; R$ be the i_{th} output from the fully connected layer. Then $y_i \; in \; R$ is computed as follows:

$$y_i = \sigma(x_1 \cdot w_1 + x_2 \cdot w_2 + \ldots + x_m \cdot w_m) \tag{3.1}$$

Here, σ is a nonlinear function (σ as the sigmoid function), and the w_i are learnable parameters in the network. The full output y is given by:

$$y_i = \sigma(x_1 \cdot w_1 + x_2 \cdot w_2 + \ldots + x_m \cdot w_m) + b \tag{3.2}$$

where b is the bias. The nodes in fully connected networks are commonly referred to as "neurons." Where every node is connected to every other nodes in the next stage.

3.3 Training

A supervised neural network can be presented as a black box with two methods such as learn and predict as following:

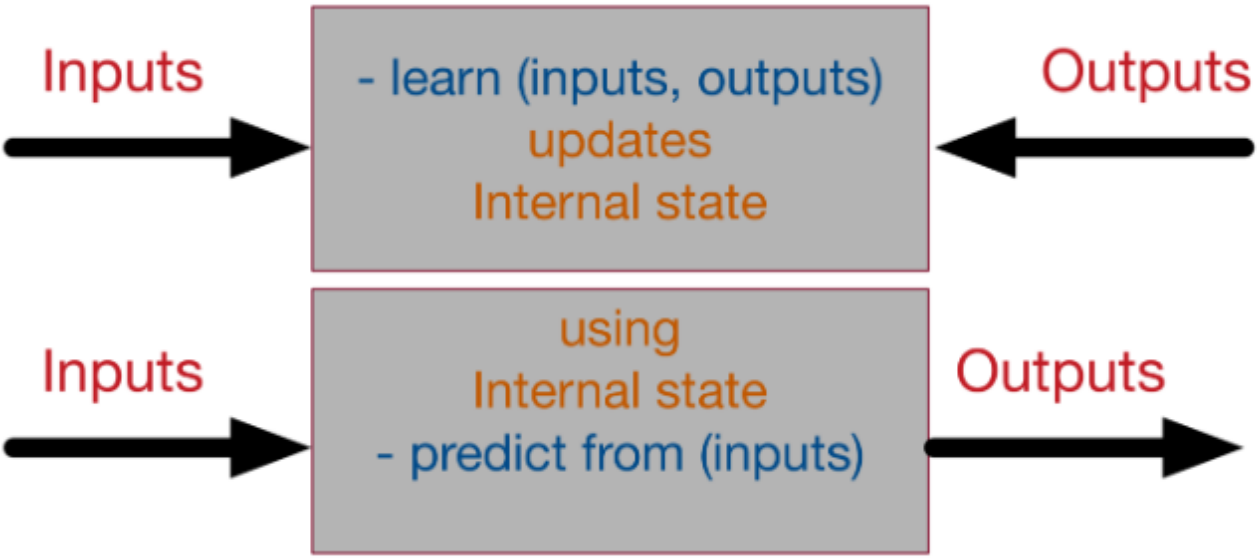

Figure 3.3: Neural network as a black box.

The learning process takes the input and as per the desired outputs it updates the parameter accordingly, so the output obtained is as close as possible from the desired output. After the learning process is done the model is ready to predict the output as per its past training experience. In order to achieve training it is divided into several processes.

3.4 Backpropagation

Back-propagation is the essence of neural network training. It is the method of fine-tuning the weights of a neural net based on the error rate obtained in the previous epoch (i.e. iteration). Proper tuning of the weights allows to reduce error rates and to make the model reliable by increasing its generalization.

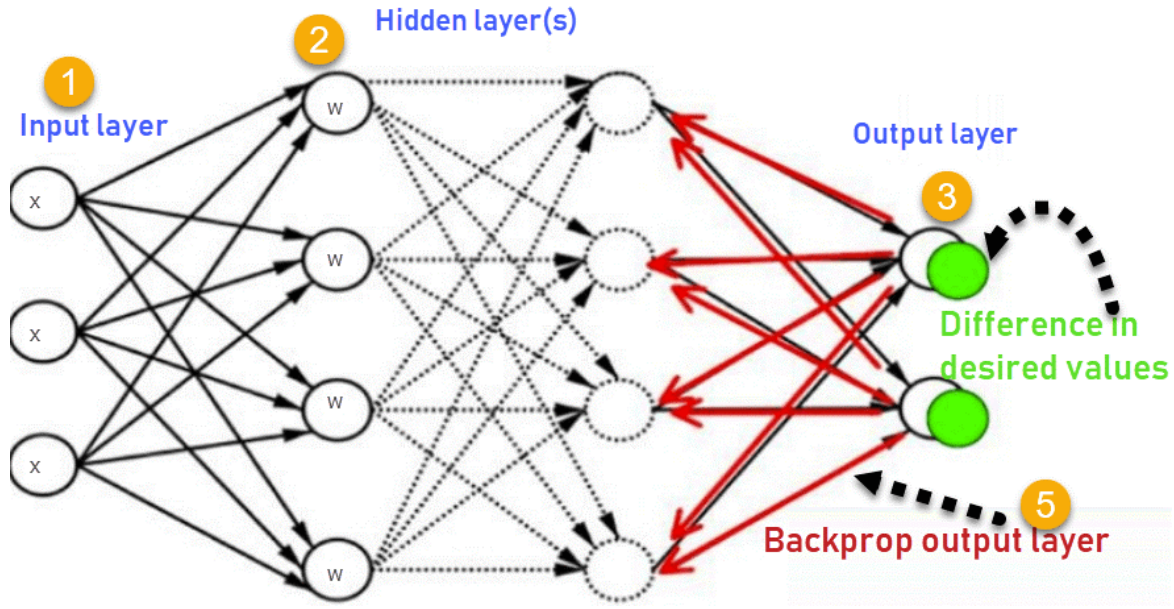

Figure 3.4: Backpropagation.

3.5 Loss Functions

Loss functions are used to evaluate how accurately our model has performed. If the prediction deviates more from the actual value, then the loss function will give high value. To produce a good prediction our model should produce low loss i.e. low deviation from an actual value. There are some optimization techniques to reduce loss such as gradient descent. We can't use loss function randomly because some loss function is sensitive which may produce some additional error. Hence it is necessary to know about loss function before using it in our model for calculating the loss in our prediction.

Difference between a Loss Function and a Cost Function:

The loss function is used for loss of single training example whereas cost function is used as the average loss of the entire training example. Hence optimization techniques are used to reduce the cost function.

Categorical Cross Entropy:

Categorical crossentropy will compare the distribution of the predictions with the true distribution, where the probability of the true class is set to 1 and 0 for the other classes. the true class is represented as a one-hot encoded vector, the closer the model's outputs are to that of the original vector, the lower the loss. Refer Eq. 3.3.

$$\mathcal{L}(y, \hat{y}) = -\sum_{j=1}^{M}\sum_{i=1}^{N}(y_{ij}\log(\hat{y}_{ij})) \tag{3.3}$$

where $\hat{y}$ is the predicted value. It is a Softmax activation and Cross-Entropy loss. using this loss, we can train a CNN to output a probability. It is used for multi-class classification. The CE Loss with Softmax activations is:

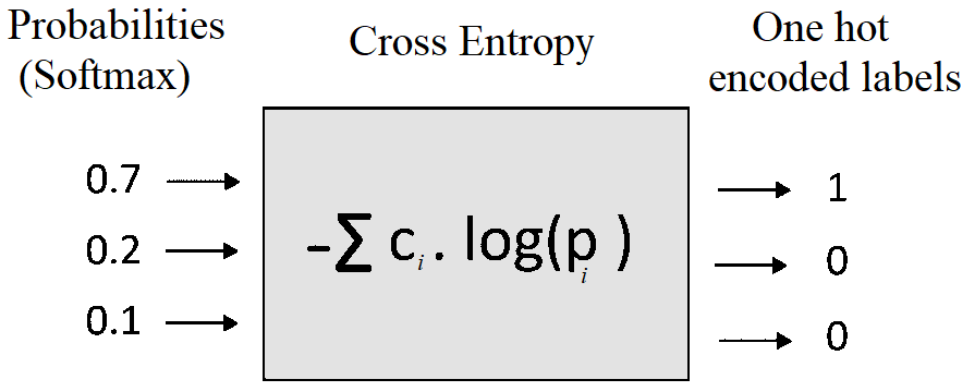

Figure 3.5: Hard decision from softmax probailities.

$$CategoricalLoss = \left(\frac{-1}{M}\right)\sum_{p=1}^{M}\log\left(\frac{e^{S_p}}{\sum_{j=1}^{C}e^{S_j}}\right) \tag{3.4}$$

Where each S_p in M is the CNN score for each positive class.

Binary Cross Entropy:

Binary crossentropy is a loss function used on problems involving yes/no (binary) decisions. For instance, in multi-label problems, where an example can belong to multiple classes at the same time, the model tries to decide for each class whether the example belongs to that class or not. Refer Eq. 3.5

$$\mathcal{L}(y, \hat{y}) = -\sum_{j=1}^{M}\sum_{i=1}^{N}(y_j \log(P(y_j)) - (1 - y_j)\log(1 - P(y_j))) \tag{3.5}$$

3.6 Gradient Descent

Gradient descent is an optimization algorithm that is used to minimize some function by continuously moving towards the minimal value of cost function. We use gradient descent to update the parameters of our machine learning model.

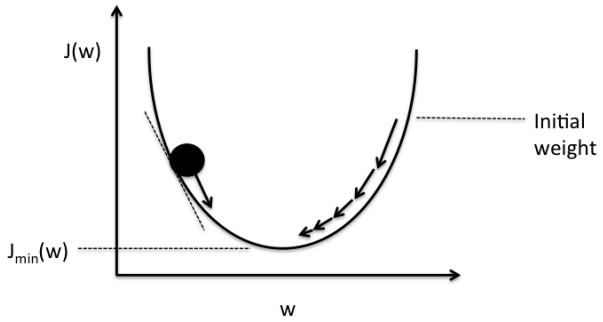

Figure 3.6: Schematic of Gradient Descent.

As shown in Fig. 3.6, error *J(w)* is a function of internal parameters of the model i.e. weights and bias. The current error is typically propagated backward to a previous layer, where it is used to modify the weights and bias in such a way that the error is minimized. The arrow is shown in Fig. 3.6 refers to the size of these steps is termed as the learning rate. Gradient descent is an iterative updating process continues until convergence, and the variable vector w achieved at convergence will be outputted as the (globally or locally) optimal variablefor the deep learning models. The pseudo-code of the vanilla gradient descent algorithm is available in Algorithm 1.

Algorithm 1 Gradient descent algorithm [5]

Require: Training set $\mathcal{T}$; Learning rate α; Normal distribution std: σ.
Ensure: Model parameter w
  Initialize parameter with Normal distribution $w \sim \mathcal{N}(0, \sigma^2)$
  Initialize convergence tag = *False*
  while tag == *False* **do**
    Compute gradient $\nabla_w J(w; \mathcal{T})$ on the training set $\mathcal{T}$
    Update variable $w = w$ - $\alpha \cdot \nabla_w J(w; \mathcal{T})$)
    if convergence condition holds **then**
      tag = *True*
    end if
  end while
  return model variable w

3.7 Activation Functions

Activation functions acts as a gate between input feeding and the output going to the next layer. It decides whether a neuron should be activated or not for the next layer. It does the non-linear task to the input so it can learn and perform more complex operations. As Activation function is a differentiable non linear function back propagation is possible as the gradient along with errors are passed to previous layers to update the weights and bias It is used to obtain the output in a range using different activation functions such as -1 to 1 or 0 to 1 depending upon the output needed.

Sigmoid:

Sigmoid function is a smooth function and can be continuously differentiable bounds the output in the range of 0 to 1. It is used for the models where the output is in the form of probabilities as probability lies in the range 0 to 1. The small change in input would bring a large change in output, so sigmoid pushes the output towards its extreme which is the desirable quality. Refer Fig. 3.7 and Eq. 3.6

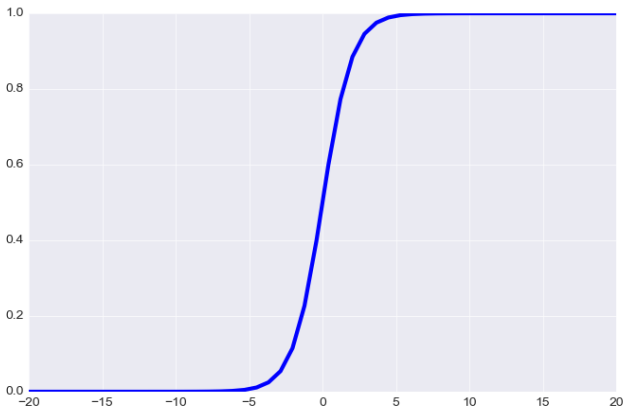

Figure 3.7: Sigmoid function.

$$\sigma(z) = \frac{1}{1 + \exp(-z)} \tag{3.6}$$

ReLU:

Rectified Linear Unit (ReLU) is computationally efficient, it allows the network to converge quickly. It looks like a linear function but it has derivative function which is useful for the back propagation. *R(z)* is zero when z is less than zero and *R(z)* is equal to z when z is above or equal to zero. All negative values become zero which in turn affects the result by not mapping negative values accurately. Refer Fig. 3.8 and Eq. 3.7

$$R(z) = max(0, z) \tag{3.7}$$

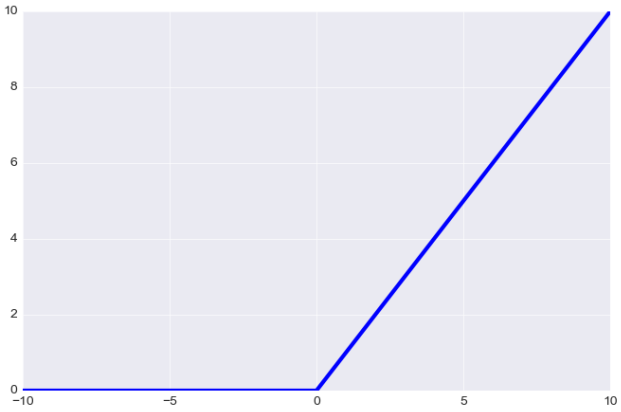

Figure 3.8: ReLU function.

Softmax:

Softmax is used to handle classification problem where there are more than 2 classes. It normalizes the output for each class in the range 0 to 1, and divides by their sum from which a probability is obtained which is used for the classification. For example consider [2.0, 1.0, 0.1], when we apply the softmax function we would get [0.7, 0.2, 0.2]. So now we can use these as probabilities for the value to be in each class. Refer Eq. 3.8

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \qquad for \;\; i, j = 1, 2, \ldots, K \tag{3.8}$$

3.8 Optimization

Optimization algorithms help to minimize the error function. The error function is a function that is dependent on the model's learning parameters from which the model predicts the output. The learnable parameters of a model are responsible for effective training and to produce accurate results. To update these parameters different optimizers are used such as Adam, SGD, etc.

ADAM:

Adam optimizer is an adaptive learning rate optimizer. Adam is composed of RMSprop and Stochastic Gradient Descent with momentum. It uses squared gradients to scale the learning rate like RMSprop and also it takes advantage of momentum by using moving average of the gradient instead of gradient itself like SGD with momentum. Adam is an adaptive learning rate method, it computes individual learning rates for different parameters. Adam uses estimations of first and second moments of gradient to adapt the learning rate for each weight of the neural network. The first moment is mean, and the second moment is uncentered variance i.e. the mean is not subtracted while taking gradient. Adam is evaluated by using exponential moving averages, computed on the gradient

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{3.9}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) {g_t}^2 \tag{3.10}$$

Moving averages of gradient and squared gradient. Where m and v are moving averages, g is gradient ,and beta is new introduced hyper-parameters of the algorithm. They have default values of 0.9 and 0.999 respectively. for different values of t the gradient changes we can generalise this as

$$m_t = (1 - \beta_1) \sum_{i=0}^{t} {\beta_1}^{t-i} g_i \tag{3.11}$$

Following are the steps for Adam optimization algorithm:

Algorithm 2 Adam algorithm. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise [7]

Require:

α: Stepsize

β_1, $\beta_2 \in [0,1)$: Exponential decay rates for the moment estimates

f(w): Stochastic objective function with parameters w

w_0: Initial parameter vector

$m_0 \leftarrow 0$ (Initialize 1^{st} moment vector)

$v_0 \leftarrow 0$ (Initialize 2nd moment vector)

$t_0 \leftarrow 0$ (Initialize timestep)

while w_t not converged **do**

$t \leftarrow t + 1$

$g_t \leftarrow \nabla_w f_t(w_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep t)

$m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)

$v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)

$\hat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)

$\hat{v}_t \leftarrow m_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)

$w_t \leftarrow w_{t-1} - \alpha \cdot \hat{m}_t / (\sqrt{\hat{v}_t} + \epsilon)$ (Update parameters)

end while

return w_t (resulting parameters)

Chapter 4

Proposed Methodology

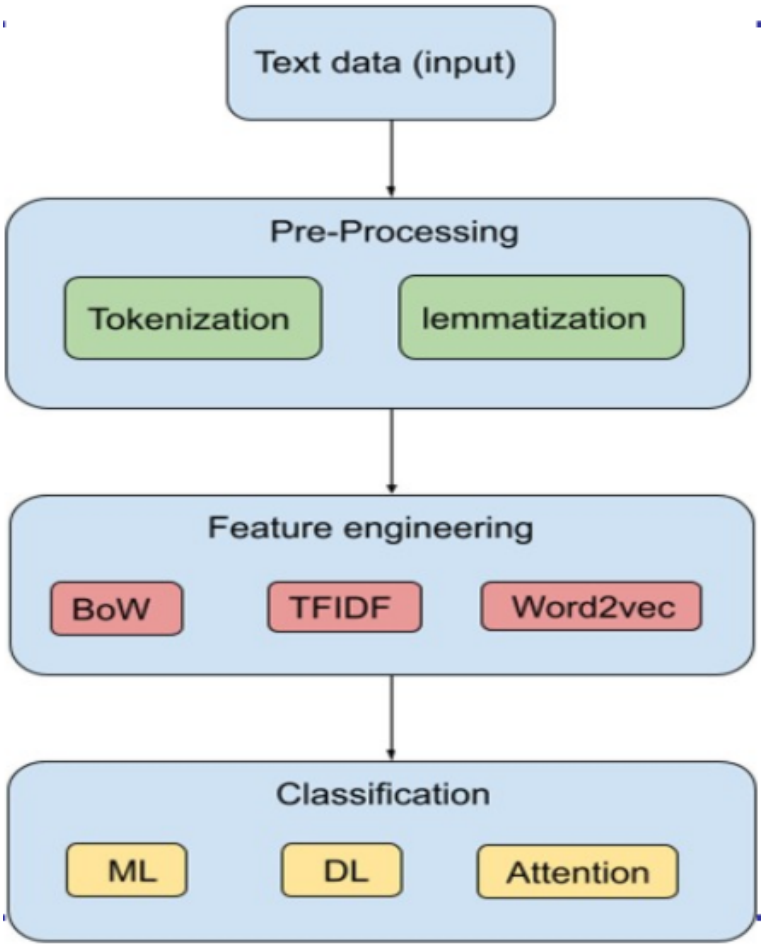

Figure 4.1: Flowchart

Here, we will provide input dataset then we will do pre-processing of the input by doing tokenization and lemmatization then we will apply feature engineering on the input data i.e Bag Of Words, TFIDF, and word2vec after processing the input data we will finally classify the dataset by using attention mechanism which comes under Deep learning which is a subset of machine learning.

4.1 Text Preprocessing :

Tokenization:

The tokenization refers to splitting up a larger body of text into smaller lines, words, or even creating words for a non-English language.

Example: text = “Hello everyone welcome to our Project” after tokenization : [‘Hello’ , ‘everyone’ , ‘welcome’ , ‘to’ , ‘our’, ‘Project’] text = ”Mumbai is a great city! Isn’t it?” after tokenization : [‘Mumbai’, ‘is’, ‘a’, ‘great’, ‘city’, ‘!’, ‘Is’, “n’t”, ‘it’, ‘?’]

Here, by using nltk which stands for natural language toolkit. In the above all the words in the sentence are getting separated into a list as Punkt is sentence tokenizer which separates every text in the sentence and makes a token i.e individual word and shows them in list form. tokenization is done for every word and symbol.

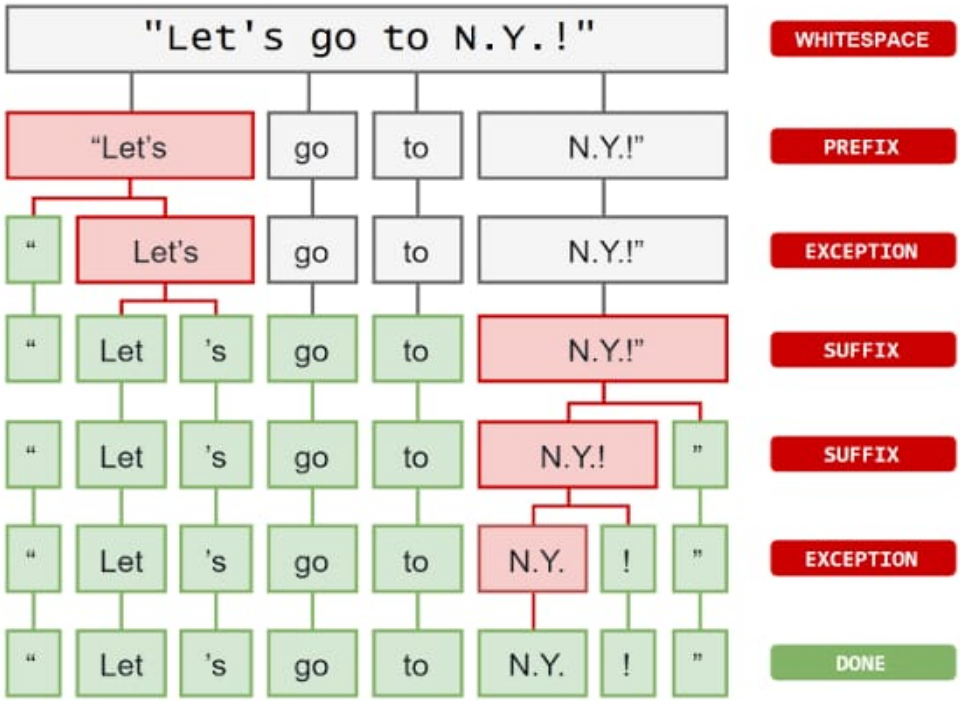

Figure 4.2: Tokenization

Lemmatization:

Text lemmatization is the process of eliminating redundant prefixes or suffixes of a word and extracting the base word (lemma). Here, Wordnet is a lexical database for the English language, it is used to find the meaning of words, synonyms, antonyms, word relation, grammers, etc. Lemmatizer is eliminating the extra suffixes and prefix i.e "s" is the suffix which is getting eliminated as lemmatizer itself helps to convert the given word into its root form which is available in the wordnet package. In lemmatization basically the word is converted to the root verb and it tries to reduce the word to its lowest form so it would occupy less memory. For eg. The lemma for studying and studying will be studying itself.

The goal of both stemming and lemmatization is to reduce inflectional forms and sometimes derivationally related forms of a word to a common base form.However, the two words differ in their flavor. Stemming usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. Lemmatization usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma . If confronted with the token saw, stemming might return just s, whereas lemmatization would attempt to return either see or saw depending on whether the use of the token was as a verb or a noun. The two may also differ in that stemming most commonly collapses derivationally related words, whereas lemmatization commonly only collapses the different inflectional forms of a lemma. Linguistic processing for stemming or lemmatization is often done by an additional plug-in component to the indexing process, and a number of such components exist, both commercial and open-source.

Pre-processing :

Converting sentence to lowercase and removing stopwords like (is, a). This is done because the maximum value of information lies in the key words. The data sets used in machine learning are large and may contain a vocabulary of lakhs and millions of words hence pre processing is preferred.
Model vocabulary:
welcome, machine, learning, now, start, good, practice Vocabulary = 7 words.
Hence
Scoring for Eg1 = [1,1, 2,1,1,0,0]
Scoring for Eg2 = [0, 0,1,0,0,1,1] These values are decided based on their occurrence in the context.

4.2 Feature engineering

Vectorization:

Vectorization is the process of converting words into numbers. We are converting all the words in the sentence into numerical form.
N - gram : It is a neighbouring sequence of items (words, letters or symbols) in a document. An N-gram model is made by counting the occurrence of word sequences occurring in sample texts and then estimating the probabilities. It is a type of probabilistic model which is trained in sample texts. This model is useful in many NLP applications such as speech recognition machine translation and predicting text inputs. types of n-grams are unigram, bigram, trigram.
For example, Our project is text classification.
unigram : [‘our’, ‘project’, ‘is’, ‘text’ , ‘classification’]
bigram : [‘our project’, ‘project is’, ‘is text’ , ‘text classification’]
trigram : [‘Our project is’, ‘project is text’ , ‘ is text classification’]
Lets understand the need of probability by taking an example:
1. Thank you so much for your help.
2. I really appreciate your help.
3. Excuse me, do you know what time it is?
4. I’m really sorry for not inviting you.
5. I really like your watch.
Now suppose we write “I really like your garden” and consider it as a bigram model so the model will learn the occurrence of every two words, to determine the probability of a word occurring after a certain word.
For example, from the 2nd, 4th, and the 5th sentence in the example above, we know that after the word “really” we can see either the word “appreciate”, “sorry”, or the word “like” occurs. So the model will calculate the probability of each of these sequences.Suppose we’re calculating the probability of word “w1” occurring after the word “w2,” then the formula for this is as follows:
count(w2 w1) / count(w2)
which is the number of times the word occurs in the required sequence, divided by the number of the times the word before the expected word occurs in the corpus.
From our example sentences, let’s calculate the probability of the word “like” occurring after the word “really”: count(really like) / count(really) = 1 / 3 = 0.33.
Similarly, for the other two possibilities: count(really appreciate) / count(really) = 1 / 3 = 0.33
count(really sorry) / count(really) = 1 / 3 = 0.33
So when I type the phrase “I really,” and expect the model to suggest the next word, it’ll get the right answer only once out of three times, because the probability of the correct answer is only 1/3.

TF-IDF:

TF-IDF in NLP stands for Term Frequency – Inverse document frequency. It is a very popular topic in Natural Language Processing which generally deals with human languages. During any text processing, cleaning the text (preprocessing) is vital. Further, the cleaned data needs to be converted into a numerical format where each word is represented by a matrix (word vectors).
Formula :

$$TF_{(i,j)} = \frac{n_{(i,j)}}{\sum n_{(i,j)}} \tag{4.1}$$

where $n_{(i,j)}$=number of times nth word occurred
$\sum n_{(i,j)}$ = total number of words

$$IDF_i = log\left(1 + \frac{N_D}{f_i}\right) \tag{4.2}$$

Here the Log function is used to reduce the relevance of a phrase with a high frequency, for eg. The count of 1 million will be decreased to 19.9 using log base 2. In simpler terms if the term frequency for the word 'computer' in doc1 is 10 and 20 in doc2, we can conclude that doc2 is more relevant for the term 'computer' than doc1. However, if the term frequency of the same word, 'computer,' for doc1 is 1 million and doc2 is 2 millions, there is no significant difference in terms of relevancy because both documents have a high count for the term 'computer.'

$$w_{i,j} = tf_{i,j} \ * \ log\left(\frac{N}{df_i}\right) \tag{4.3}$$

How Tf-idf Vectorizer works:
The Tf-idf Vectorizer tokenizes documents, learns the vocabulary and inverse frequency weighting of documents, and allows you to encode new documents.
What is Tf-idf Vectorizer?
TF-IDF is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is accomplished by multiplying two metrics: the number of times a word appears in a document and the word's inverse document frequency over a set of documents.
What is Fit_transform?
Fit_transform() is used on the training data so that we can scale the training data and also learn the scaling parameters of that data.

Bag-of-words:

How bag-of-words works:
Bag-of-Words is a method of feature extraction with text data. A bag-of-words is a representation of text that describes the occurrence of words within a document. It is called a "bag" of words because any information about the order or structure of words in the document is discarded.
Why Bag-of-Words is Used:
Problems with text is that it is messy and unstructured, and machine learning algorithms prefer structured well defined fixed-length inputs . By using the Bag-of-Words technique we can convert variable-length texts into a fixed-length vector. The machine learning models work with numerical data rather than textual data, by using the bag-of-words (BoW) technique, we convert a text into its equivalent vector of numbers

Bag-of-words examples:
Eg1: Welcome to Machine Learning, now start learning
Eg2: Learning is a good practice

4.3 Attention mechanism:

The basic idea in Attention is that each time the model tries to predict an output word, it only uses parts of an input where the most relevant information is concentrated instead of an entire sentence i.e it tries to give more importance to the few input words.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right) \tag{4.4}$$

There are three separate Linear layers for the Query, Key, and Value. Each Linear layer has its own weights. The input is passed through these Linear layers to produce the Q, K, and V matrices. The Query, Key and Value inputs are generated based on their respective weighted matrices that is
image
An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Encoder Self-Attention

The input sequence is sent into the Input Embedding and Location Encoding, which creates an encoded representation for each word in the input sequence that captures its meaning and position. This is supplied into the Self-Attention in the first Encoder, which creates an encoded representation for each word in the input sequence that now includes the attention ratings for each word. Each Self-Attention module adds its own attention ratings to each word's representation as it travels through all of the Encoders in the stack.

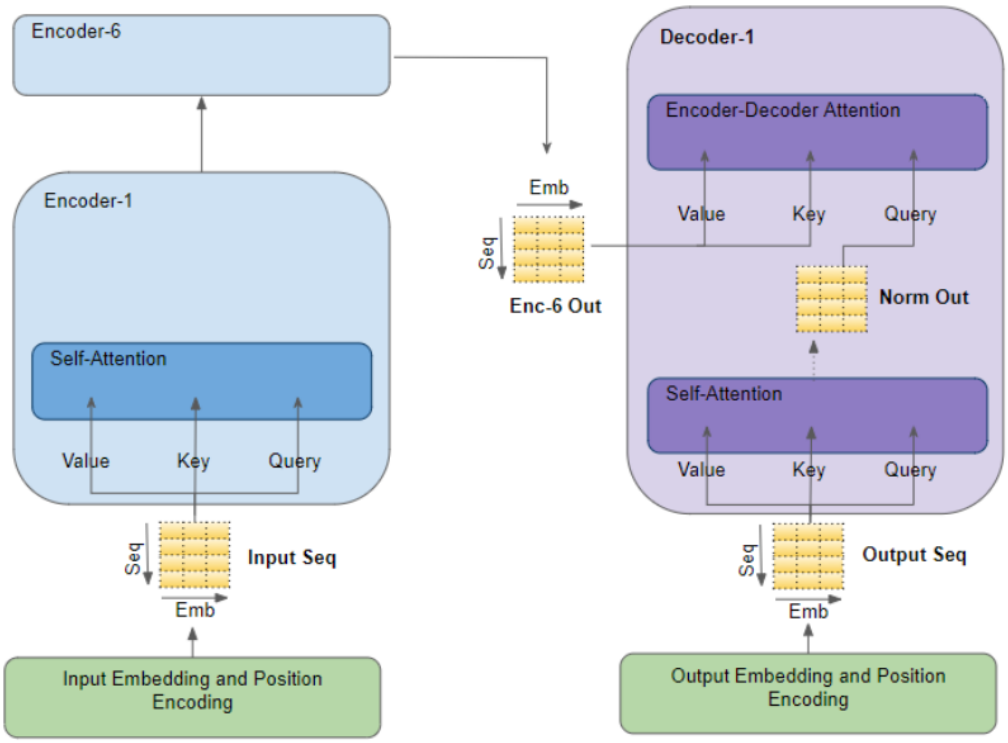

Figure 4.3: Self Attention

Decoder Self-Attention

The target sequence is given to the Output Embedding and Location Encoding in the Decoder stack, which generates an encoded representation for each word in the target sequence that captures the meaning and position of each word. This is passed into the first Decoder's Self-Attention with all three parameters, Query, Key, and Value, which creates an encoded representation for each word in the target sequence, which now includes the attention ratings for each word. This is provided to the Query parameter in the Encoder-Decoder Attention in the first Decoder after going through the Layer Norm.

The output of the last Encoder on the stack is also supplied to the Encoder-Decoder Attention's Value and Key arguments. As a result, the Encoder-Decoder Attention receives a representation of both the target sequence (from the Decoder Self-Attention) and the input sequence (from the Encoder-Decoder Attention) (from the Encoder stack). As a result, it generates a representation containing the attention scores for each target sequence word that also incorporates the effect of the input sequence's attention scores. Each Self-Attention and each Encoder-Decoder Attention add their own attention ratings to each word's representation as this traverses through all of the Decoders in the stack.

Multiple Attention Heads

The Attention module of the Transformer performs its calculations numerous times in parallel. An Attention Head is the name given to each of them. The Attention module separates its Query, Key, and Value arguments N times and sends each split to a different Head. The results of all of these comparable Attention computations are then pooled to get a final Attention score. This is known as Multi-head attention, and it allows the Transformer to encode many associations and subtleties for each word with more ease.

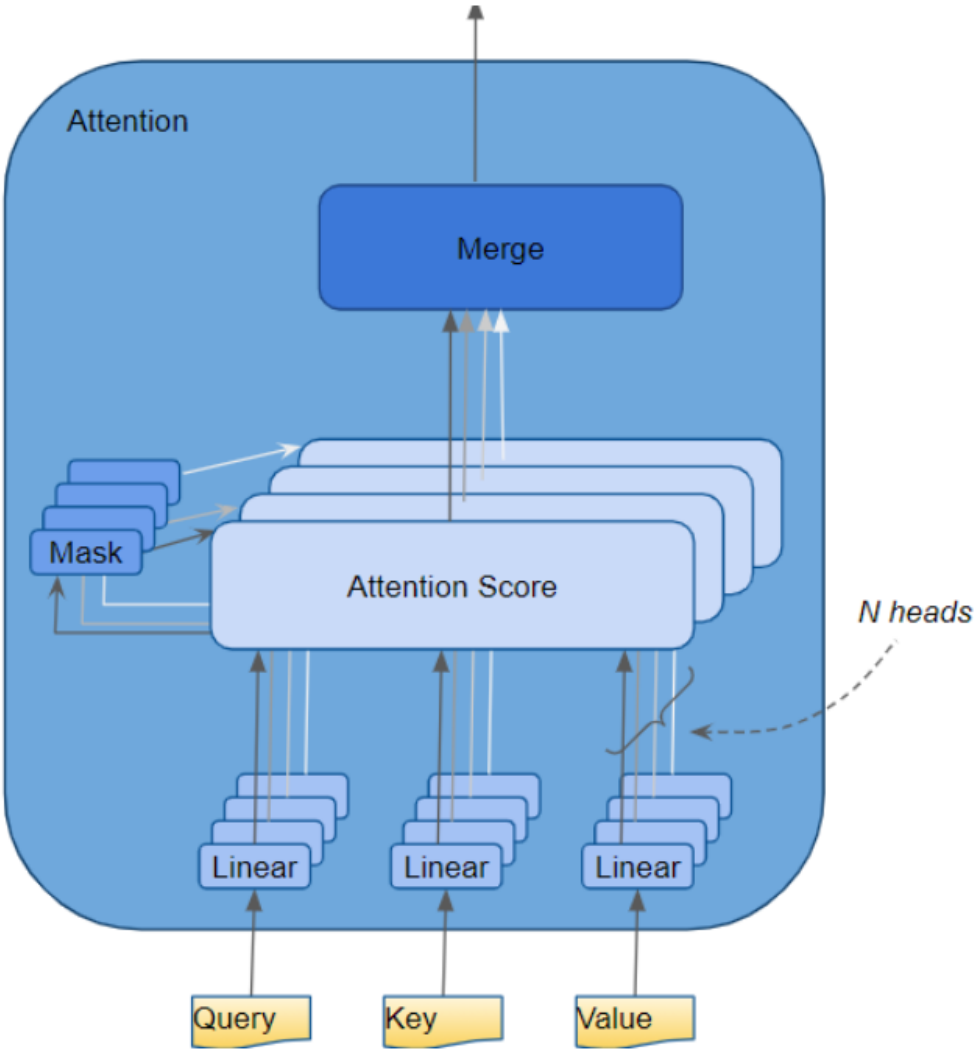

Figure 4.4: Multi-head Attention

Attention Hyperparameters:

There are three hyperparameters that determine the data dimensions:

Embedding Size — width of the embedding vector (we use a width of 6 in our example). This dimension is carried forward throughout the Transformer model and hence is sometimes referred to by other names like 'model size' etc.

Query Size (equal to Key and Value size)— the size of the weights used by three Linear layers to produce the Query, Key, and Value matrices respectively (we use a Query size of 3 in our example)

Number of Attention heads (we use 2 heads in our example)

Layers of Input The Input Embedding and Position Encoding layers provide a shape matrix (Number of Samples, Sequence Length, Embedding Size) that is supplied to the first Encoder's Query, Key, and Value.

Layers in a straight line For the Query, Key, and Value, there are three different Linear layers. The weights of each Linear layer are different. The Q, K, and V matrices are created by passing the input across these Linear layers.

Splitting data across Attention heads

The data is now distributed across the several Attention heads, allowing each to process it independently. The crucial thing to remember is that this is merely a logical separation. The Query, Key, and Value matrices are not physically separated into one for each Attention head. For the Query, Key, and Value, a single data matrix is employed, with conceptually different regions of the matrix for each Attention head. There are no distinct Linear layers for each Attention head, either. The Attention heads all work on the same Linear layer, but on their own logical segment of the data matrix. The weights of linear layers are logically partitioned by head.

Linear layer weights are logically partitioned per head This logical split is done by partitioning the input data as well as the Linear layer weights uniformly across the Attention heads.
We can achieve this by choosing the Query Size as below:

$$Query\ Size = \frac{Embedding\ Size}{Number\ of\ heads} \tag{4.5}$$

Reshaping the Q, K, and V matrices

The Linear layers' Q, K, and V matrices are rearranged to incorporate an explicit Head dimension. Each'slice' now corresponds to a head's matrix. By exchanging the Head and Sequence dimensions, this matrix is rearranged once again. The dimensions of Q are now depicted, notwithstanding the absence of the Batch dimension (Batch, Head, Sequence, Query size). Although the Q matrix is a single matrix, we may conceive of it as a conceptually independent Q matrix for each head in the final stage.

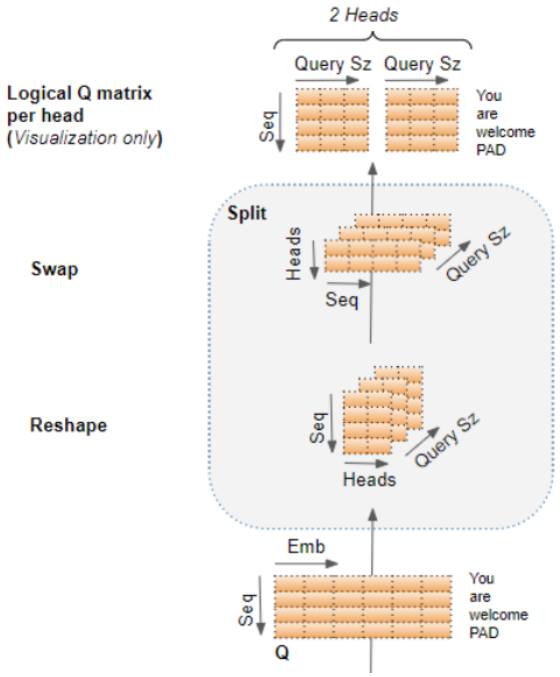

Figure 4.5: Reshaping

Compute the Attention Score for each head

The three matrices, Q, K, and V, are now spread over the heads. The Attention Score is calculated using these. We'll illustrate the computations for a single head by ignoring the first two dimensions and focusing on the last two (Sequence and Query size) (Batch and Head). In essence, we may suppose that the calculations we're looking at are'repeated' for each head and sample in the batch (although, obviously, they are happening as a single matrix operation, and not as a loop).

The first step is to multiply the two matrices Q and K. The result now has a Mask value applied to it. The mask is used in the Encoder Self-attention to mask out the Padding values so they don't contribute in the Attention Score.

In the Decoder Self-attention and the Decoder Encoder-Attention, which we'll get to later in the flow, different masks are used. The result is now scaled by dividing it by the square root of the Query size before applying a Softmax on it. Between the output of the Softmax and the V matrix, another matrix multiplication is done. The following is the whole Attention Score computation in the Encoder Self-attention

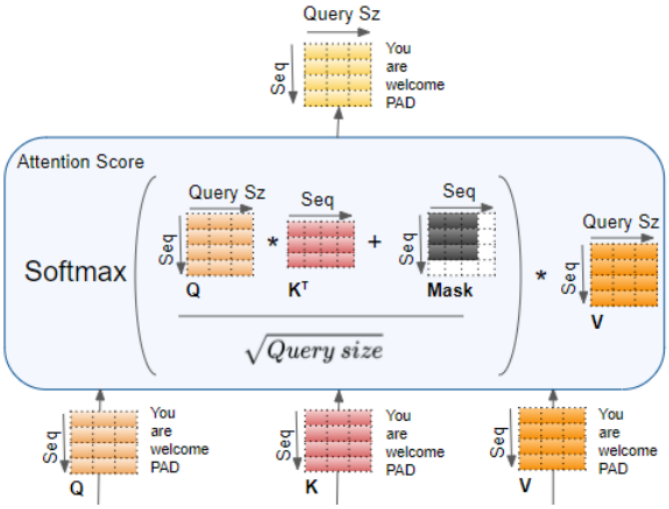

Figure 4.6: Computing Attention Scores

Merging each Head's Attention Scores together Masking

Each head now has its own Attention Score, which must be added together to get a single score. The Merge operation is similar to the Split operation in that it reverses the order of operations. Simply rearranging the output matrix to remove the Head dimension does this. The steps are as follows:

Swap the Head and Sequence dimensions to reshape the Attention Score matrix. In other words, the matrix shape changes from (Batch, Head, Sequence, Query size) to (Batch, Head, Sequence, Query size) (Batch, Sequence, Head, Query size).

Reshape the Head dimension to (Batch, Sequence, Head * Query size) to collapse it. The Attention Score vectors for each head are successfully concatenated into a single merged Attention Score.

The Decoder Self-Attention is similar to the Encoder Self-Attention in that it acts on each word of the target sequence instead of the entire sequence. In the same way, the Masking removes the Padding words from the target sequence. The Encoder-Decoder Attention gets its information from two different places. Unlike Encoder Self-Attention, which calculates the interaction between each input word and other input words, and Decoder Self-Attention, which calculates the interaction between each target word and other target words, Encoder-Decoder Attention calculates the interaction between each target word and each input word. As a consequence, each cell in the resultant Attention Score represents the interaction between one Q (target sequence word) and all other K (input sequence) words, as well as all V (input sequence) words.

Why Attention mechanism?:

When we think of the English term "attention," we think of focusing your attention on something and paying closer attention. Long sequences are a challenge in Machine Translation, and they are also an issue in most other NLP applications. As needed, the process allows the model to focus and place more "Attention" on the necessary elements of the input sequence.

4.4 Transformer Architecture:

Transformer architecture excels in handling text data which is inherently sequential. It uses self attention relating every word in input sequence to every other word. In the fig.4.7 on the left is the encoder model, and on the right, the decoder model, both have a core block that is repeated N times: "an attention and a feed-forward network."The Figure 4.7 depicts the multi-head Attention and the main architecture. You can see that there are multiple Attention heads arising from different V, K, Q vectors, and they are concatenated:

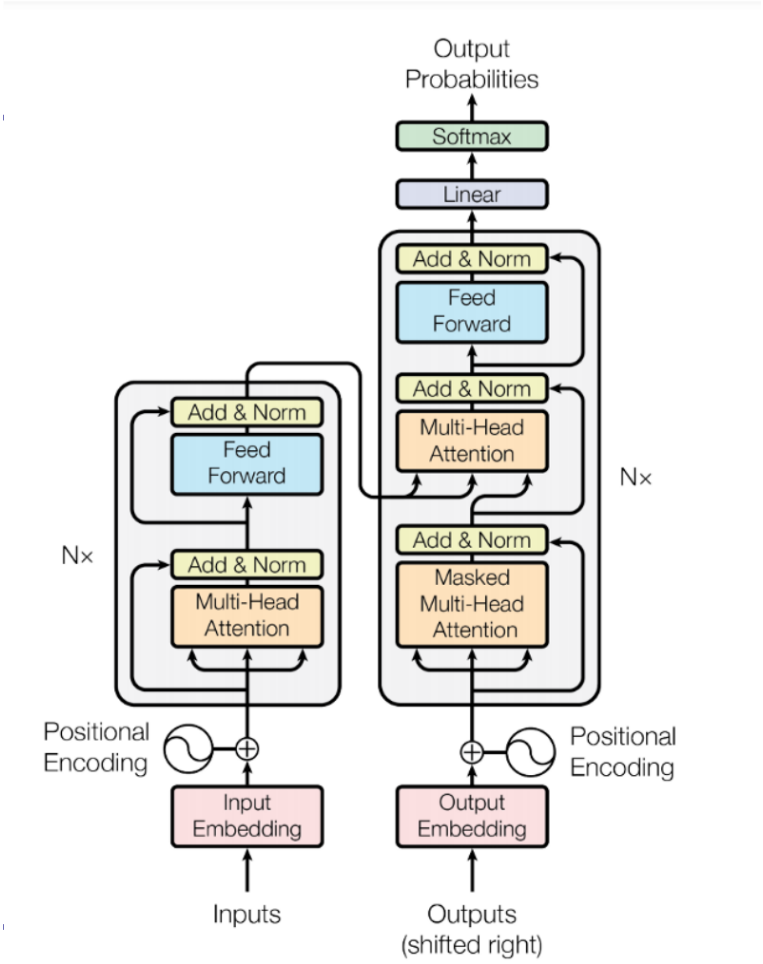

Figure 4.7: Transformer Model Architecture

4.4.1 Embedding & Positional Encoding:

The embedding layer encodes the meaning of a word (Transformer has 2 Embedding Layer) Input Sequence is fed to the 1st embedding layer through input embedding which is on the left side fig.4.7 and Target Sequence is fed to the 2nd embedding layer through output embedding which is on the right side of fig. 4.7, after shifting targets right by one position and inserting a start token in 1st place.

Embeddings make it easier to do machine learning on large inputs like sparse vectors representing words. Ideally, an embedding captures some of the semantics of the input by placing semantically similar inputs close together in the embedding space. An embedding can be learned and reused across models.The major advantage of this mechanism over RNN is that the sequence is Input in parallel, but the positional information is lost has to be added back separately. There are 2 positional encoding layers.

$$PE_{pos,2i} = sin\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{4.6}$$

$$PE_{pos,2i+1} = cos\left(\frac{pos}{1000^{\frac{2i}{d_{model}}}}\right) \tag{4.7}$$

4.4.2 Model Computations

Let's start with how to compute self-attention with vectors, then go on to how it's actually done — with matrices.

The first step in calculating self-attention is to divide each of the encoder's input vectors into three vectors (in this case, the embedding of each word). So we make a Query vector, a Key vector, and a Value vector for each word. The embedding is multiplied by three matrices that we trained throughout the training procedure to get these vectors.

The dimensions of these new vectors are less than the embedding vector. The embedding and encoder input/output vectors have a dimensionality of 64, whereas the embedding and encoder input/output vectors have a size of 512. They don't have to be smaller; this is an architecture option to keep multiheaded attention computations (basically) constant.

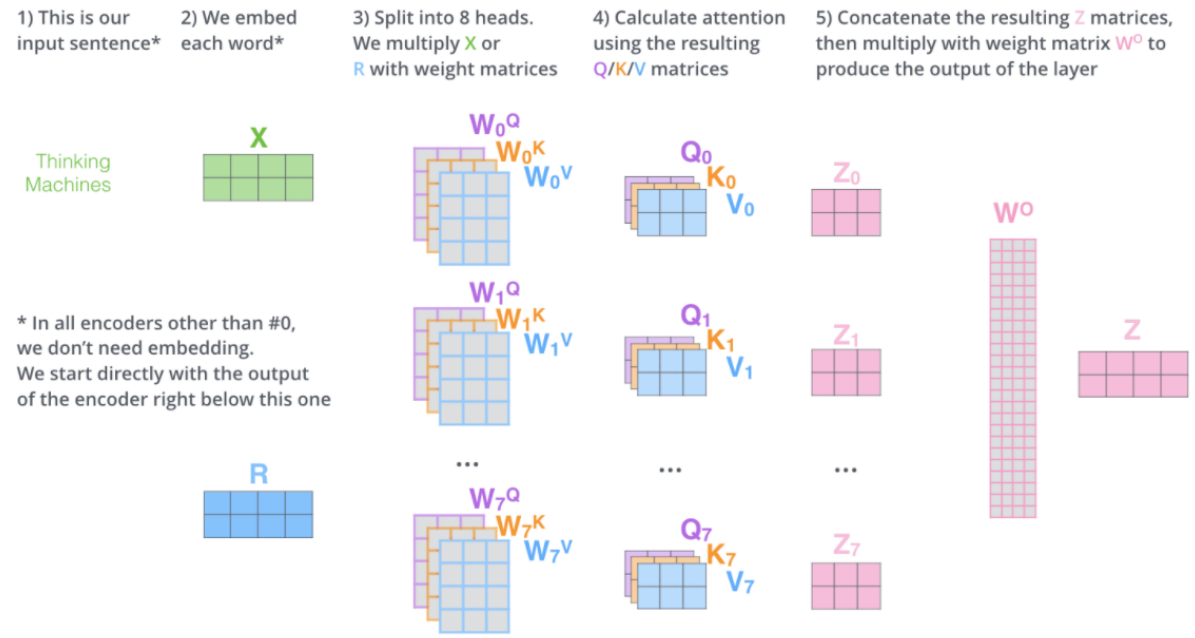

Figure 4.8: Model Computations

For calculating and thinking about attention, the "query," "key," and "value" vectors are important. After reading the section below on how attention is calculated, you'll know pretty much everything there is to know about the role each of these vectors plays.

Calculating a score is the second stage in calculating self-attention. Let's say we're trying to figure out how much self-attention we have for the first word in this example, "Thinking." Each word in the input sentence must be compared to this term. As we encode a word at a certain location, the score dictates how much emphasis to place on other portions of the input text.

The score is calculated by taking the dot product of the query vector with the key vector of the respective word we're scoring. So if we're processing the self-attention for the word in position 1, the first score would be the dot product of q1 and k1. The second score would be the dot product of q1 and k2.

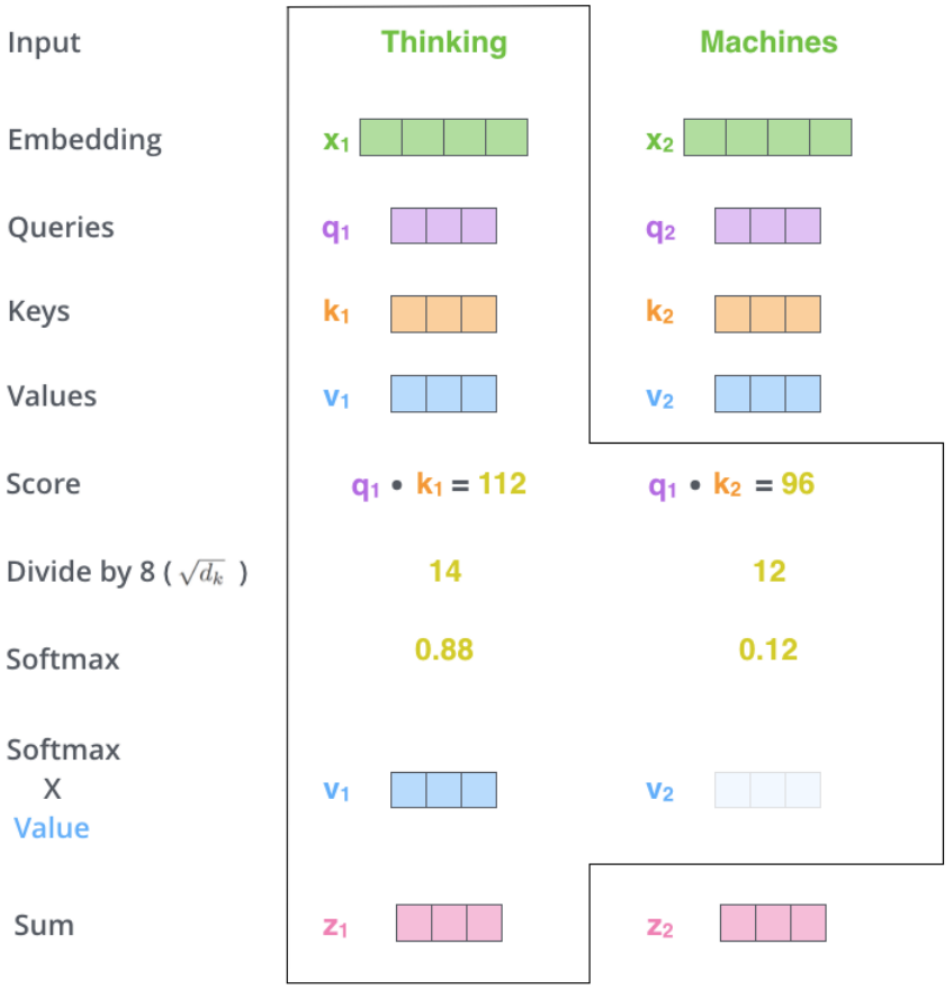

Figure 4.9: Model Computations

Divide the scores by 8 (the square root of the dimension of the key vectors used in the study - 64) in the third and fourth phases. As a result, gradients become more stable. Other values may be available, but this is the default), and then run the result via a softmax operation. Softmax makes the scores all positive and adds them up to one. This softmax score decides how much each word in this place will be conveyed. Obviously, the word in this place has the greatest softmax score, but it's occasionally helpful to focus on another word that is related to the present term.

The softmax score is multiplied by each value vector in the fifth phase (in preparation to sum them up). The idea is to maintain the values of the word(s) we wish to focus on intact while drowning out unnecessary ones (by multiplying them by tiny numbers like 0.001, for example).

The weighted value vectors are added together in the sixth phase. At this point, the output of the self-attention layer is produced (for the first word).

The self-attention calculation is now complete. We may now transmit the generated vector to the feed-forward neural network. However, for quicker processing, this computation is done in matrix form in the real implementation. Now that we've seen how the computation works on a word level.

4.4.3 SoftMax:

The softmax function, also known as softargmax or normalized exponential function, is a generalization of the logistic function to multiple dimensions. It is used in multinomial logistic regression and is often used as the last activation function of a neural network to normalize the output of a network to a probability distribution over predicted output classes, based on Luce's choice axiom.

The softmax function takes as input a vector z of K real numbers, and normalizes it into a probability distribution consisting of K probabilities proportional to the exponentials of the input numbers. That is, prior to applying softmax, some vector components could be negative, or greater than one; and might not sum to 1; but after applying softmax, each component will be in the interval (0,1), and the components will add up to 1, so that they can be interpreted as probabilities. Furthermore, the larger input components will correspond to larger probabilities.

The SoftMax regression is a form of logistic regression that normalizes an input value into a vector of values that follows a probability distribution whose total sums up to 1. It is used to get the attention weights.

$$\sigma(\overline{Z})i = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_i}} \tag{4.8}$$

4.5 Dataset:

We have taken IMDb (Internet Movie Database) which is an online database that includes cast, production crew, and personal biographies, story summaries, trivia, ratings, and fan and critical reviews for films, television shows, home videos, video games, and streaming entertainment online. Message boards, another fan service, was decommissioned in February 2017. Originally a fan-run website, IMDb.com, Inc., an Amazon subsidiary, now owns and operates the database.
The IMDb dataset includes 50,000 questionnaires, allowing for about 30 audits per film. It was created by Stanford University academics Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts in 2011. The data was divided into training and test sets in an equitable distribution.
This is a dataset of movie reviews from IMDB, labeled by sentiment (positive/negative). A negative review receives a $\leq$4 out of 10 rating, while a good survey get a $\geq$ 7 out of 10 rating. This dataset does not include any neutral reviews. Reviews have been preprocessed, and each review is encoded as a list of word indexes (integers). For convenience, words are indexed by overall frequency in the dataset, so that for instance the integer "3" encodes the 3rd most frequent word in the data. This allows for quick filtering operations such as: "only consider the top 10,000 most common words, but eliminate the top 20 most common words". As a convention, "0" does not stand for a specific word, but instead is used to encode any unknown word[4].
The researchers gathered the raw data from the IMDb website. They looked through the content information in each review to see if there were any highlights that might be used to determine whether the review was good or negative. The reviews were then separated into training and test sets and uploaded to their website in an even distribution. There are two further directories representing pos and neg tags in each of the sets' directories, allowing the information to be partitioned using various markings.

4.5.1 Exploratory Data Analysis (EDA) :

EDA is a data exploration technique to understand the various aspects of the data. The main object of EDA is to filter the data from redundancies. the data should be clean and should not contain any missing values and not even null values. In this analysis, we have to make sure that we identify the important variables in the data set and remove all unnecessary noise in the data that may actually hinder the accuracy of our conclusions when we work on model building and we must understand the relationship between the variables through EDA. we must be able to derive conclusions on gathering insights about the data for conclusive interpretation in order to move on to more complex processes in the data processing lifecycle steps involve in EDA :

1. Understand the data
2. Clean the data
3. Analysis of relationship between variables.

4.5.2 Results and Discussion :

In this project, we have done text classification with the help of an attention mechanism that included the Transformer architecture, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed and self-attention. For text classification tasks, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers.

Table 4.1: classification report

	precision	recall	f1-score	support
0	0.84	0.87	0.85	4993
1	0.86	0.85	0.85	5007
accuracy			0.85	10000
macro avg	0.85	0.85	0.85	10000
weighted avg	0.85	0.85	0.85	10000

We explore the performances of Transformer model. We use the accuracy, recall, precision, and F1 score as performance metrics.The expressions of these metrics are given as following:

$$Accuracy = \frac{True\ Positive + True\ Negative}{True\ Positive + True\ Negative + False\ Positive + False\ Negative} \tag{4.9}$$

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{4.10}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{4.11}$$

$$F1\ score = 2\frac{precision\ \ recall}{precision\ +\ recall} \tag{4.12}$$

```
Model: "text_att_bi_rnn"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        multiple                  2500000
_________________________________________________________________
bidirectional (Bidirectional multiple                  234496
_________________________________________________________________
attention (Attention)        multiple                  384
_________________________________________________________________
dense (Dense)                multiple                  257
=================================================================
Total params: 2,735,137
Trainable params: 2,735,137
Non-trainable params: 0
_________________________________________________________________
```

Figure 4.10: model summary

Validation Test:

- **It had some bad parts like the storyline although the actors performed really well and that is why overall I enjoyed it.**
  Probability of Positive: [0.98785883]
- **this movie is the worst movie.**
  Probability of Positive: [0.00688329]
- **I can watch this movie forever just because of the beauty in its cinematography.**
  Probability of Positive: [0.9762399]
- **I really enjoyed this film.**
  Probability of Positive: [0.9593219]

Chapter 5

Conclusion

In this work, we studied the Transformer, the first sequence transduction model based entirely on attention, replacing the recurrent layers most commonly used in encoder-decoder architectures with multi-headed self-attention. For Text-classification, the Transformer can be trained significantly faster than architectures based on recurrent or convolutional layers. Experimental results demonstrate that Transformer architecture with attention mechanism performs significantly better than methods described in literature survey. Attention layers of Transformer model is effective in picking out important words from the input sentences. So We can conclude that attention mechanisms are the most efficient way for text classification.The model trained on the test data gave a decent accuracy of around 85%.Additionally, we can increase the accuracy by training the model with more number of epochs.

Bibliography

[1] Zhang, Z., Zhang, S., Chen, E., Wang, X., Cheng, H., *"TextCC: New Feed Forward Neural Network for Classifying Documents Instantly. In: Wang, J., Liao, XF., Yi, Z. (eds) Advances in Neural Networks – ISNN 2005"*. ISNN 2005. Lecture Notes in Computer Science, vol 3497. Springer, Berlin, Heidelberg. Link:
`https://doi.org/10.1007/11427445`$_3$7

[2] A. B. Dieng, C. Wang, J. Gao, and J. Paisley, *"Topicrnn: A recurrent neural network with long-range semantic dependency,"*, Published as a conference paper at ICLR 2017, Link:
`https://arxiv.org/abs/1611.01702`

[3] Y. Kim, *"Convolutional neural networks for sentence classification"*, Proceedings of 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP),pp. 1746-1751, 2014. Link:
`https://aclanthology.org/D14-1181.pdf`

[4] D. Bahdanau, K. Cho, and Y. Bengio, href*"Neural machine translation by jointly learning to align and translate,"*, Published as a conference paper at ICLR 2015. Link:
`https://arxiv.org/pdf/1409.0473.pdf`

[5] Liang Yao, Chengsheng Mao, Yuan Luo, *"Graph Convolutional Networks for Text Classification,"*, Published in The Thirty-Third AAAI Conference on Artificial Intelligence (AAAI-19). Link:
`https://arxiv.org/abs/1809.05679`

[6] Vaswani Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin, *"Attention is all you need,"*, In Advances in neural information processing systems, pp. 5998-6008. 2017. Link:
`https://arxiv.org/pdf/1706.03762.pdf`

[7] K. S. Tai, R. Socher, and C. D. Manning, *"Improved semantic representations from tree-structured long short-term memory networks,"*, Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. Link:
`https://arxiv.org/pdf/1503.00075.pdf`

[8] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky , Ilya Sutskever, Ruslan Salakhutdinov, *"Dropout: A Simple Way to Prevent Neural Networks from Overfitting"*, Journal of Machine Learning Research, published 6/14. Link:
`https://www.cs.toronto.edu/ hinton/absps/JMLRdropout.pdf`

[9] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, *"Hierarchical attention networks for document classification,"*, " in Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies, 2016, pp. 1480–1489. Link:
https://www.cs.cmu.edu/ ./hovy/papers/16HLT-hierarchical-attention-networks.pdf

[10] M.-T. Luong, H. Pham, and C. D. Manning, *"Effective approaches to attention-based neural machine translation,"*, Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015. Link:
https://arxiv.org/pdf/1508.04025.pdf

[11] S. Wang, M. Huang, and Z. Deng, *"Densely connected cnn with multi-scale feature attention for text classification."*,in IJCAI, 2018, pp. 4468–4474. Link:
https://www.ijcai.org/proceedings/2018/0621.pdf

[12] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, *"Fasttext. zip: Compressing text classification models"*,paper at ICLR 2017 Link:
https://arxiv.org/pdf/1612.03651.pdf

[13] Q. Le and T. Mikolov *"Distributed representations of sentences and documents"*, in International conference on machine learning, 2014, pp. 1188–1196. Link:
https://arxiv.org/abs/1405.4053

[14] X. Zhu, P. Sobihani, and H. Guo, *"Long short-term memory over recursive structures"*, in International Conference on Machine Learning, 2015 pp. 1604–1612.

[15] J. Cheng, L. Dong, and M. Lapata *"Long short-term memory-networks for machine reading"*, pp. 1188–1196. Link:
https://arxiv.org/abs/1601.06733

[16] P. Liu, X. Qiu, and X. Huang, *"Recurrent neural network for text classification with multi-task learning,"*, Link:
https://arxiv.org/abs/1605.05101

[17] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu, *"Text classification improved by integrating bidirectional lstm with two-dimensional max pooling"*, published in NIPS 2015. Link:
https://arxiv.org/pdf/1611.06639.pdf

[18] R. Johnson and T. Zhang *"Supervised and semi-supervised text categorization using lstm for region embeddings"*, Link:
https://arxiv.org/abs/1602.02373

[19] J. Liu, W. C. Chang, Y. Wu, and Y. Yang, *""Deep learning for extreme multi-label text classification"*, in SIGIR 2017 - Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, 2017. Link:
http://nyc.lti.cs.cmu.edu/yiming/Publications/jliu-sigir17.pdf